Sami Batuhan Basmaz Ölmez
1801042653
Computer Organization HW4 Report

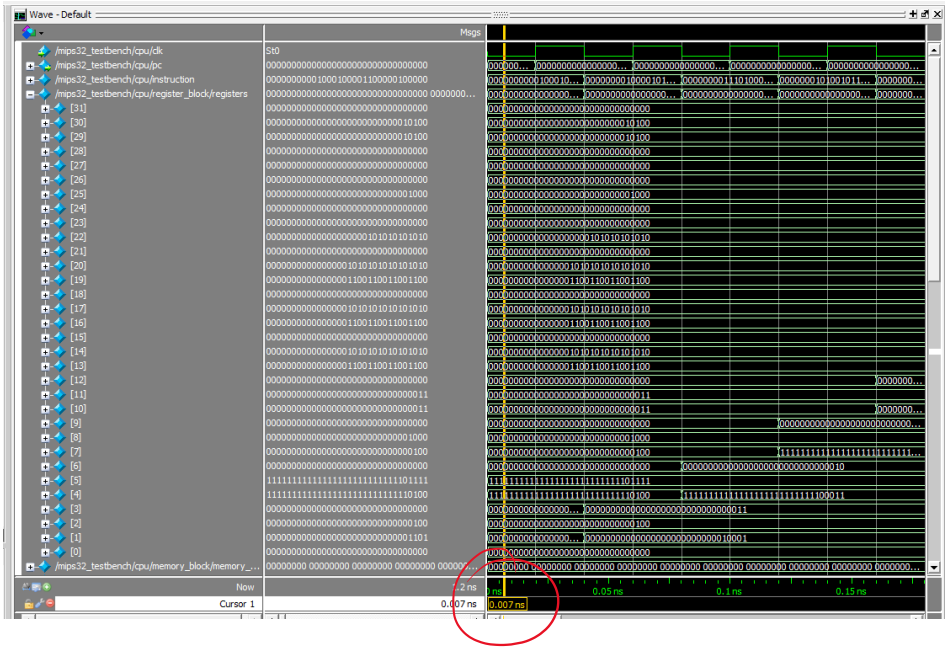
Before

Instruction: 000000 00001 00010 00011 00000 100000

addn $3, $1, $2

Rd = 3, Rs = 1, Rt = 2

Initial Decimal Values: R[3] = 0, R[1] = 13, R[2] = 4



After

Instruction: 000000 00001 00010 00011 00000 100000

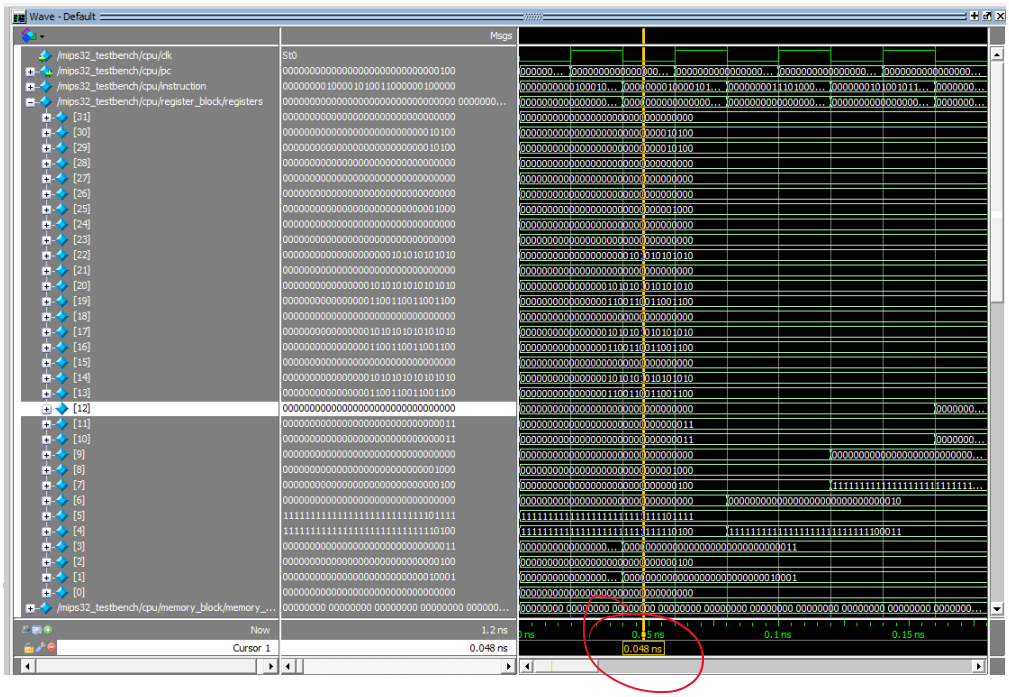
addn $3, $1, $2

Rd = 3, Rs = 1, Rt = 2

Expected Decimal Values: R[3] = 3, R[1] = 17, R[2] = 4

PASS!

0.040ns
Full cycle



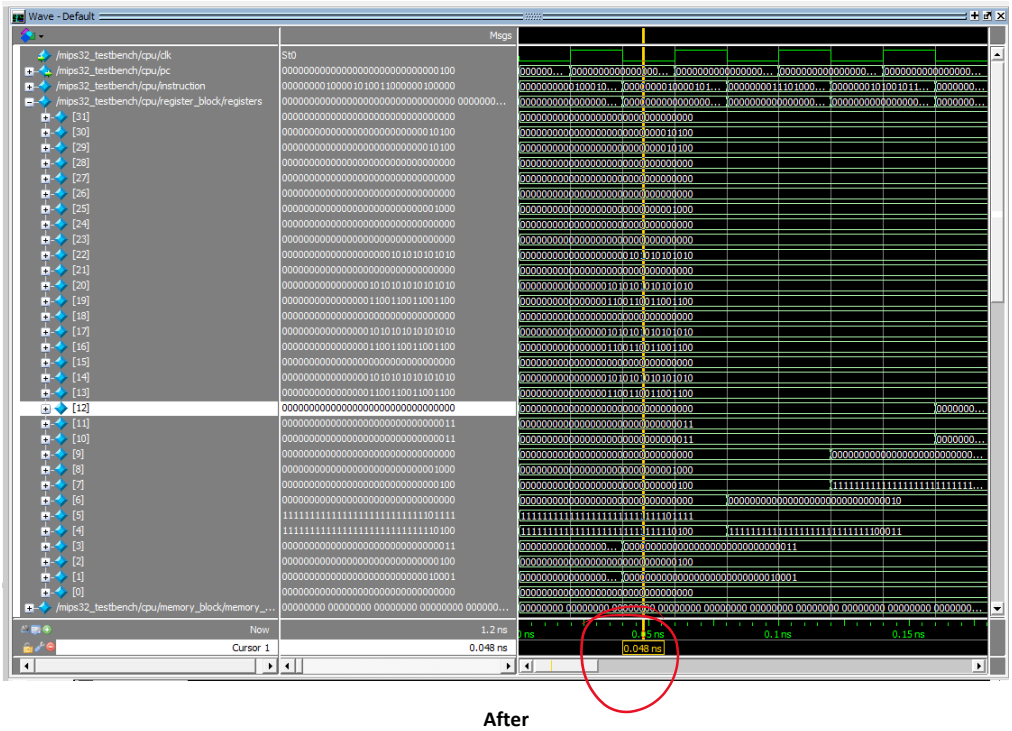
Before

Instruction: 000000 00100 00101 00110 00000 100000

addn $6, $4, $5

Rd = 6, Rs = 4, Rt = 5

Initial Decimal Values: R[6] = 0, R[4] = -12, R[5] = -17



After

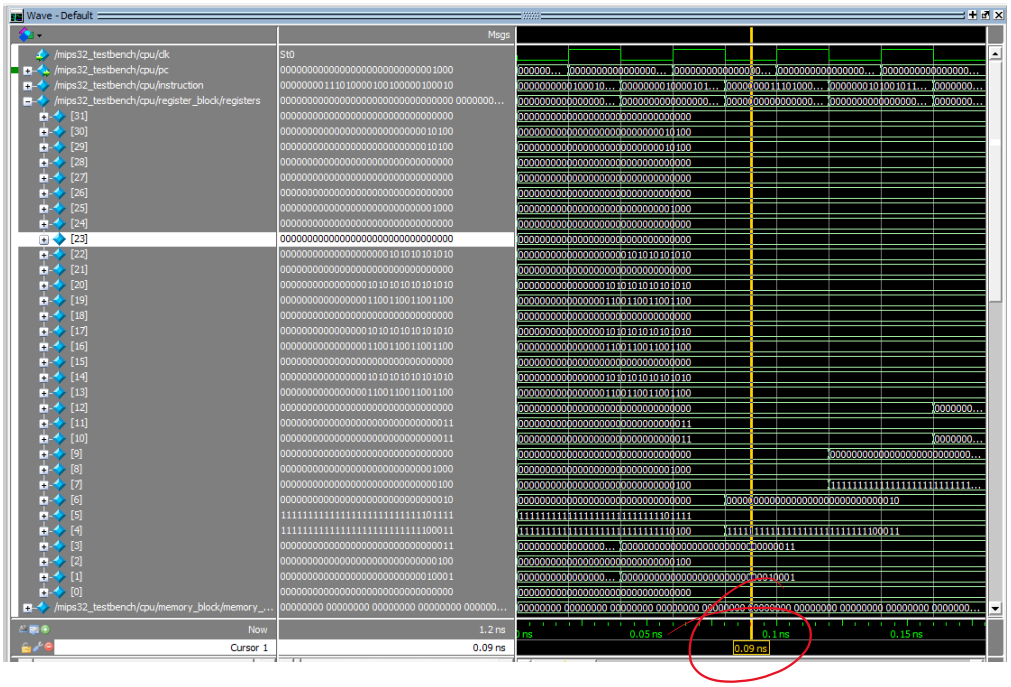
Instruction: 000000 00100 00101 00110 00000 100000

addn $6, $4, $5

Rd = 6, Rs = 4, Rt = 5

Expected Decimal Values: R[6] = 2, R[4] = -29, R[5] = -17

PASS!



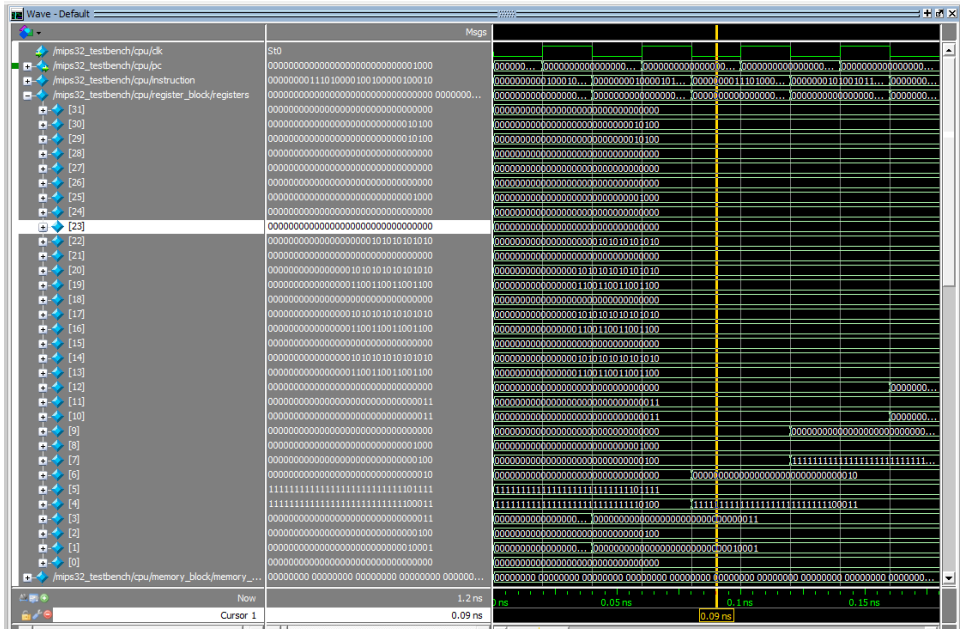
Before

Instruction: 000000 00111 01000 01001 00000 100010

subn $9 $7 $8

Rd = 9, Rs = 7, Rt = 8

Initial Decimal Values: R[9] = 0, R[7] = 4, R[8] = 8



After

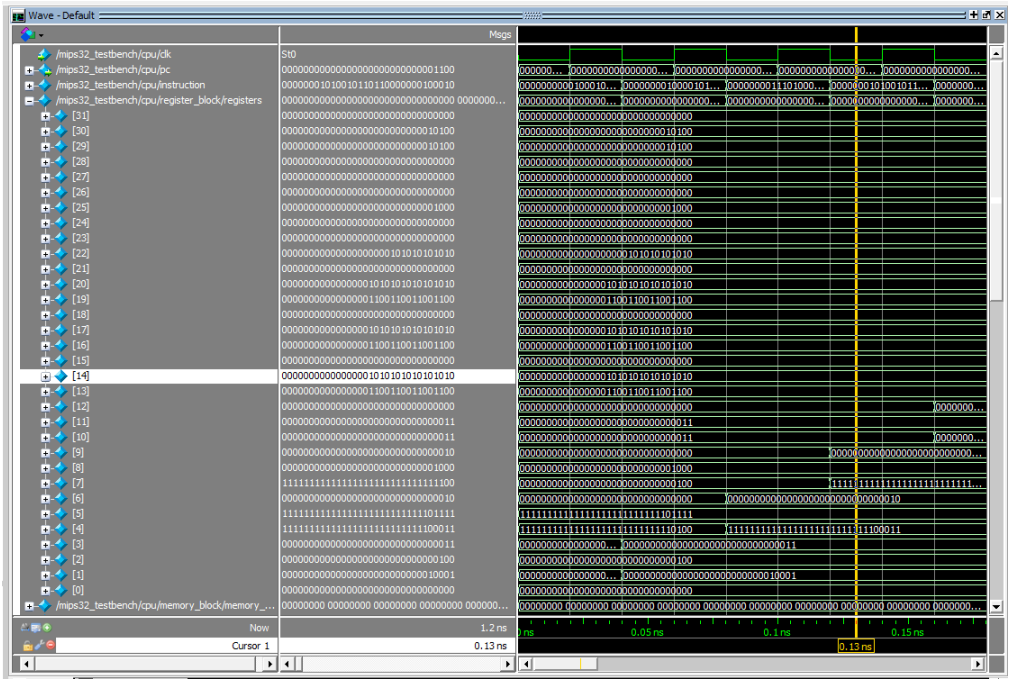
Instruction: 000000 00111 01000 01001 00000 100010

subn $9 $7 $8

Rd = 9, Rs = 7, Rt = 8

Expected Decimal Values: R[9] = 2, R[7] = -4, R[8] = 8

PASS!



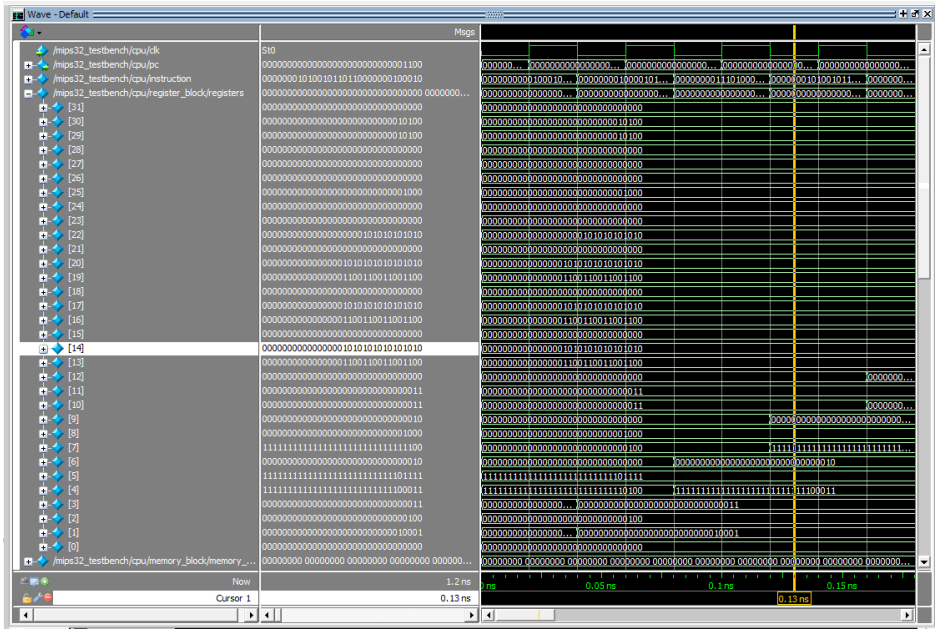
Before

Instruction: 000000 01010 01011 01100 00000 100010

subn $12 $10 $11

Rd = 12, Rs = 10, Rt = 11

Initial Decimal Values: R[12] = 0, R[10] = 3, R[11] = 3



After

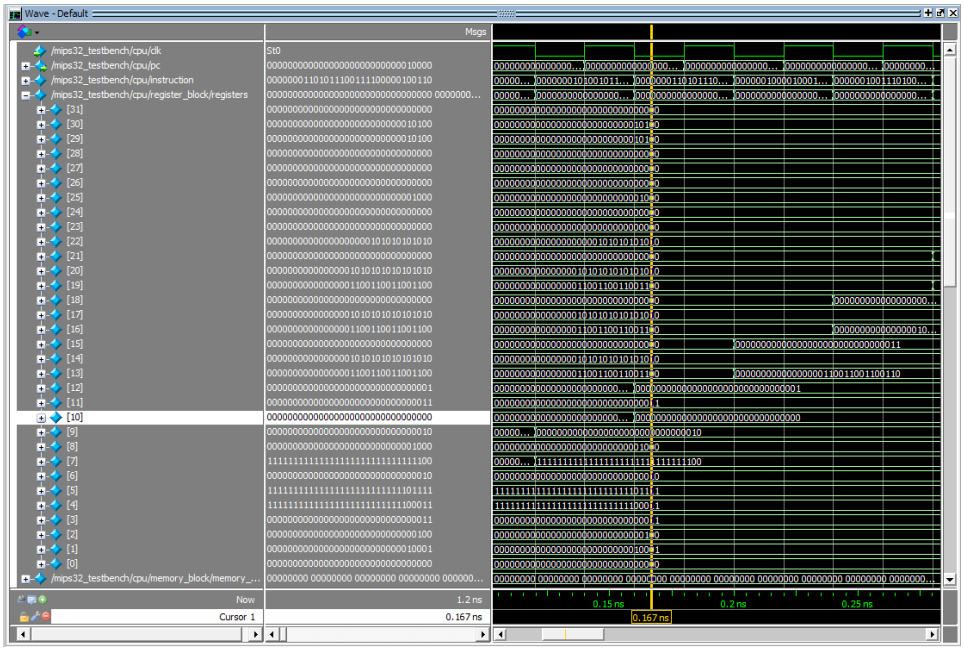
Instruction: 000000 01010 01011 01100 00000 100010

subn $12 $10 $11

Rd = 12, Rs = 10, Rt = 11

Expected Decimal Values: R[12] = 1, R[10] = 0, R[11] = 3

PASS!



Before

Instruction: 000000 01101 01110 01111 00000 100110

xorn $15 $13 $14

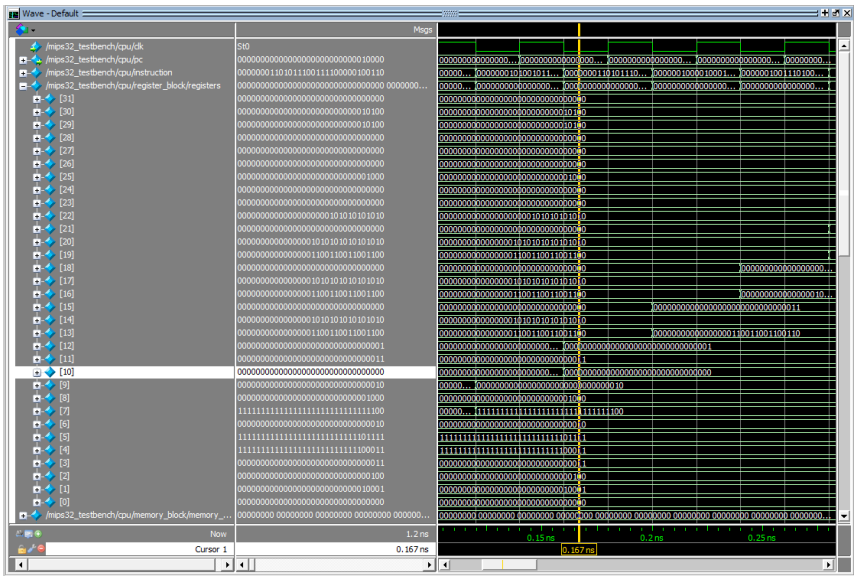
Rd = 15, Rs = 13, Rt = 14

Initial Binary Values:

R[15] = 32'b0,

R[13] = 00000000000000001100110011001100,

R[14] = 00000000000000001010101010101010,



After

Instruction: 000000 01101 01110 01111 00000 100110

xorn $15 $13 $14

Rd = 15, Rs = 13, Rt = 14

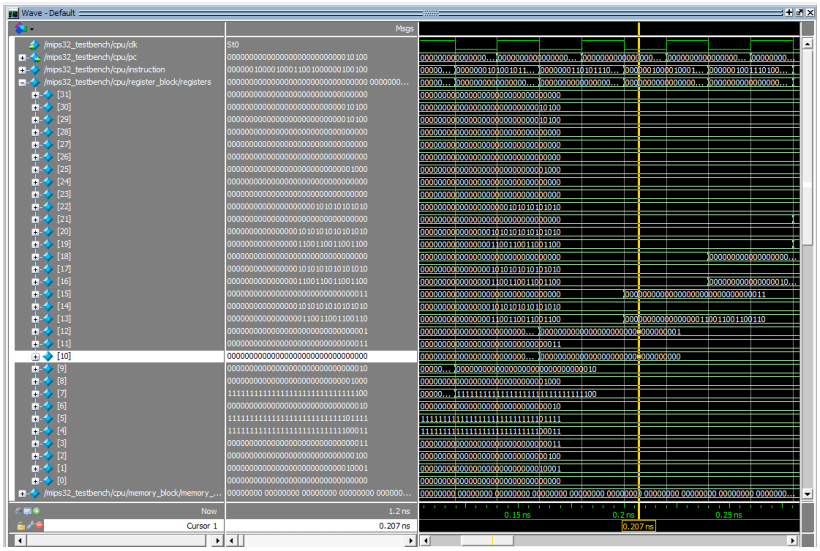
Expected Binary Values:

R[15] = 00000000000000000000000000000011,

R[13] = 00000000000000001100110011001100,

R[14] = 00000000000000001010101010101010,

PASS!



Before

Instruction: 000000 10000 10001 10010 00000 100100

andn $18 $16 $17

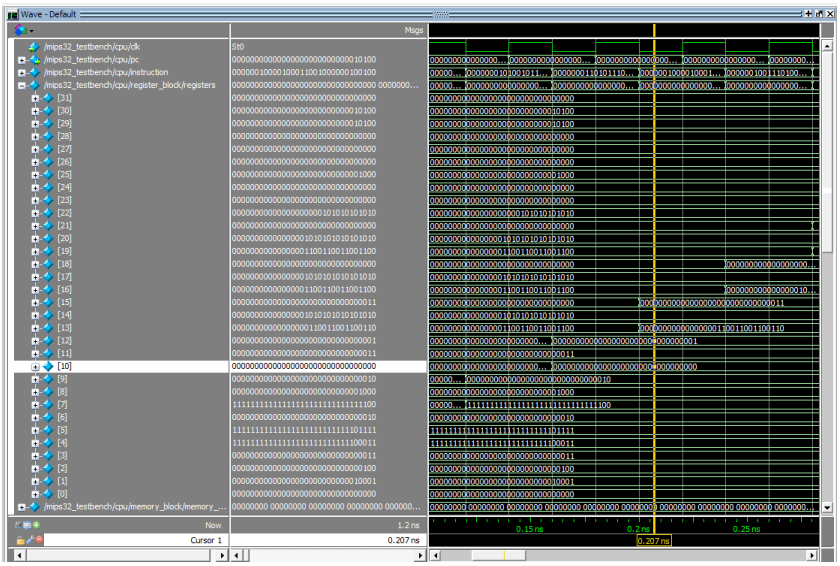
Rd = 18, Rs = 16, Rt = 17

Initial Binary Values:

R[18] = 32'b0,

R[16] = 00000000000000001100110011001100,

R[17] = 00000000000000001010101010101010,



After

Instruction: 000000 10000 10001 10010 00000 100100

andn $18 $16 $17

Rd = 18, Rs = 16, Rt = 17

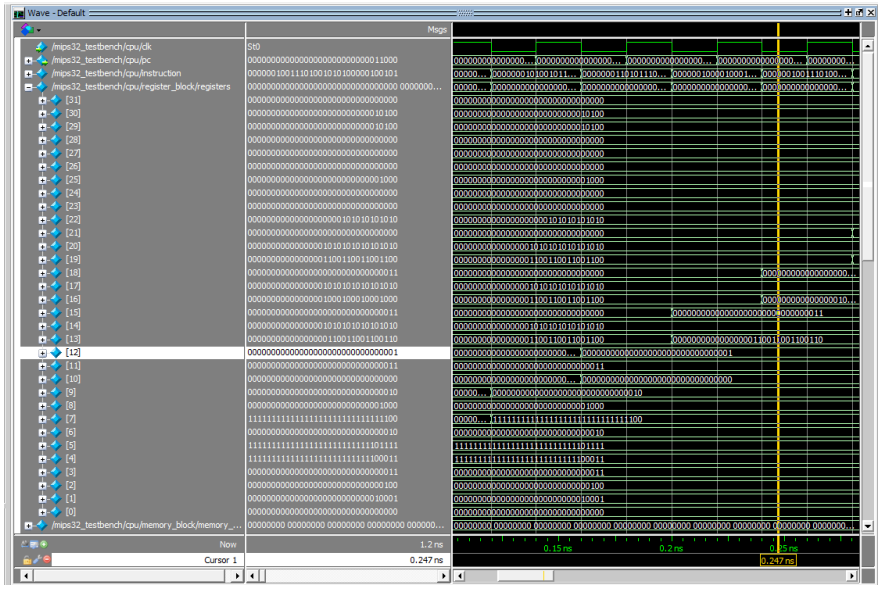
Expected Binary Values:

R[18] = 00000000000000000000000000000011,

R[16] = 00000000000000001000100010001000,

R[17] = 00000000000000001010101010101010,

PASS!



Before

Instruction: 000000 10011 10100 10101 00000 100101

orn $21 $19 $20

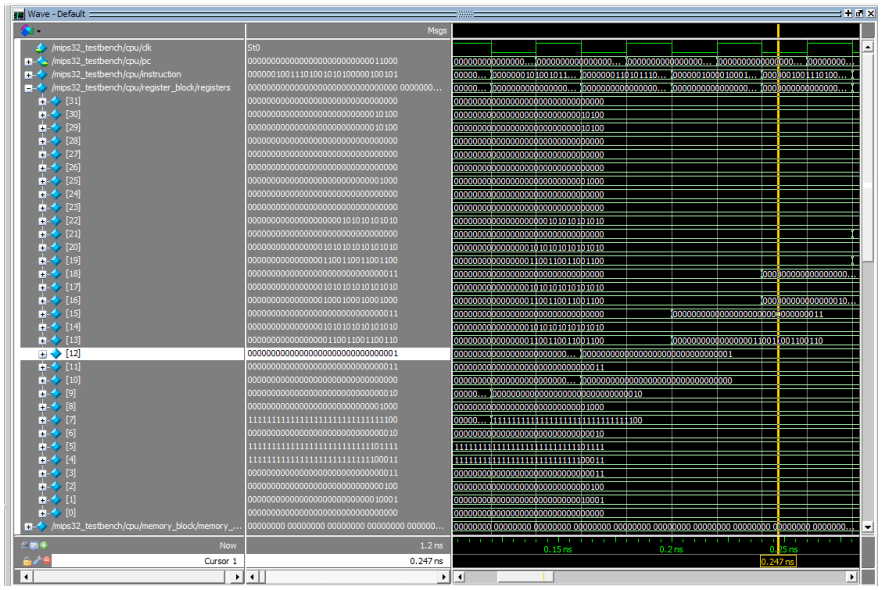
Rd = 21, Rs = 19, Rt = 20

Initial Binary Values:

R[21] = 32'b0,

R[19] = 00000000000000001100110011001100,

R[20] = 00000000000000001010101010101010,



After

Instruction: 000000 10011 10100 10101 00000 100101

orn $21 $19 $20

Rd = 21, Rs = 19, Rt = 20

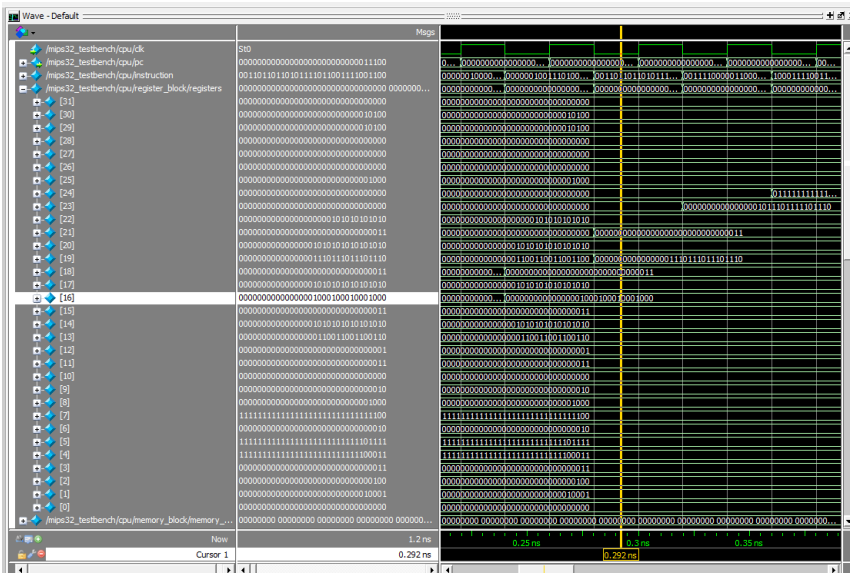
Expected Binary Values:

R[21] = 00000000000000000000000000000011,

R[19] = 00000000000000001110111011101110,

R[20] = 00000000000000001010101010101010,

PASS!



Before

Instruction: 001101 10110 10111 1011001111001100

ori $23 $22 1011001111001100

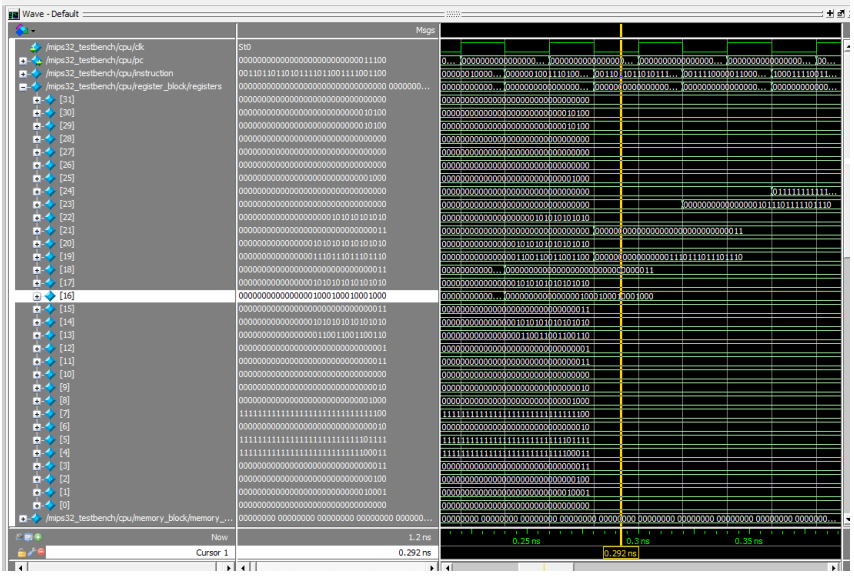
Rs = 22, Rt = 23

Initial Binary Values:

R[22] = 00000000000000000000101010101010,

R[23] = 32'b0,

Immediate: 1011001111001100



After

Instruction: 001101 10110 10111 1011001111001100

ori $23 $22 1011001111001100

Rs = 22, Rt = 23

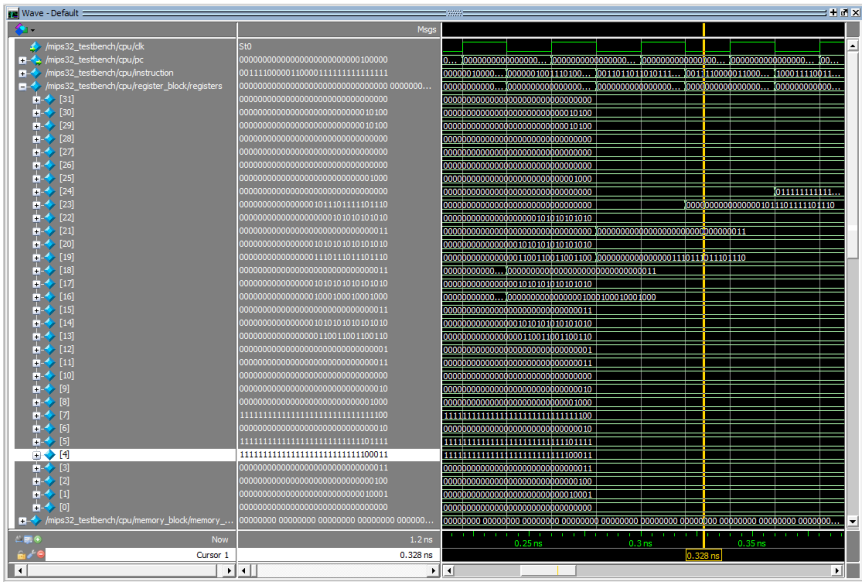
Expected Binary Values:

R[22] = 00000000000000000000101010101010,

R[23] = 00000000000000001011101111101110,

Immediate: 1011001111001100

PASS!



Before

Instruction: 001111 00000 11000 0111111111111111

lui $24 0111111111111111

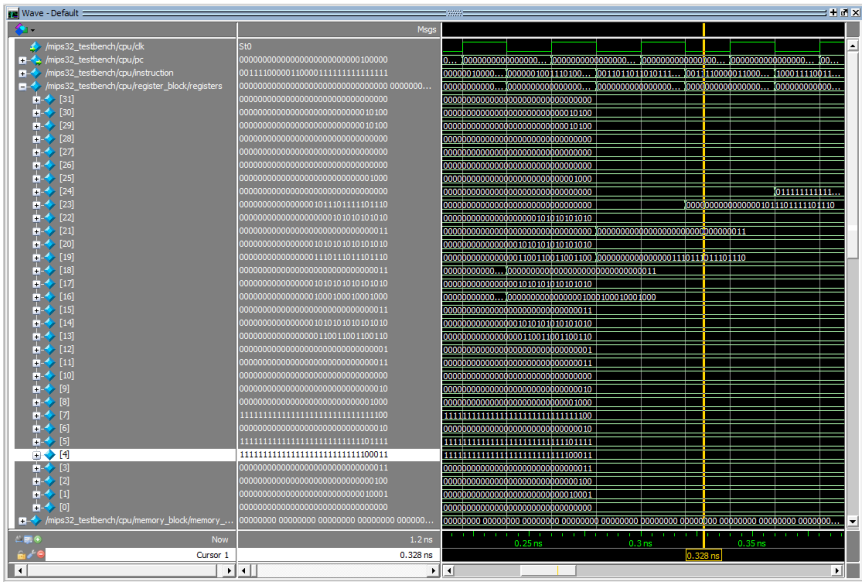
Rs = 0, Rt = 24

Initial Binary Values:

R[0] = 32'b0,

R[24] = 32'b0,

Immediate: 0111111111111111



After

Instruction: 001111 00000 11000 0111111111111111

lui $24 0111111111111111

Rs = 0, Rt = 24

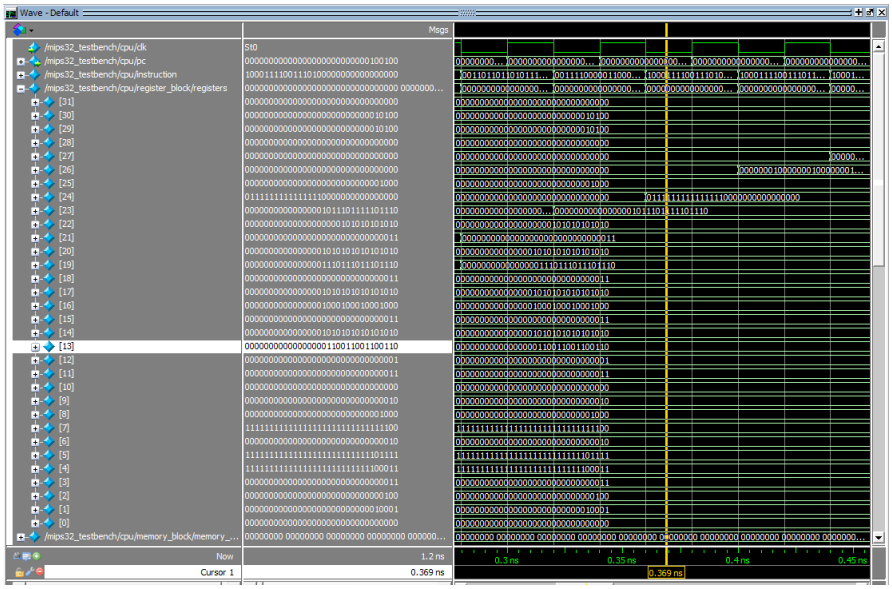
Expected Binary Values:

R[0] = 32'b0,

R[24] = 01111111111111110000000000000000,

Immediate: 0111111111111111

PASS!



Before

Instruction: 100011 11001 11010 0000000000000000

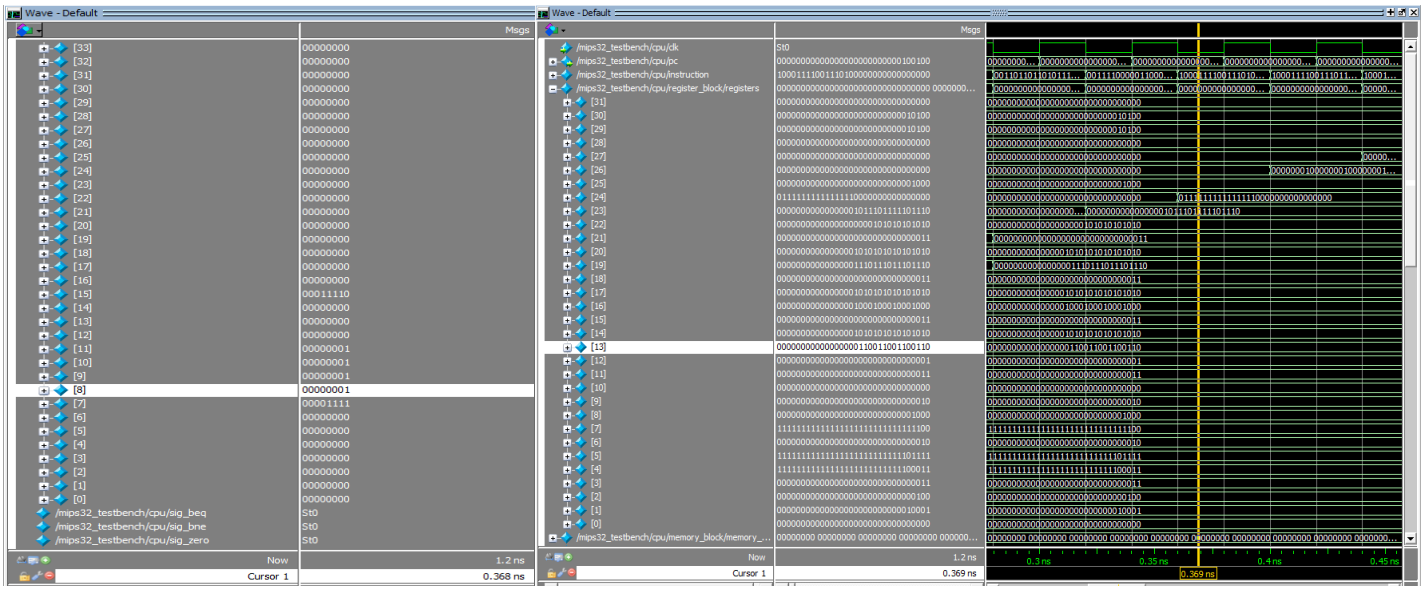
lw $26 0($25)
Rs = 25, Rt = 26

Initial Decimal Values:

R[26] = 0,

R[25] =8,

Immediate: 0000000000000000



After

Instruction: 100011 11001 11010 0000000000000000

lw $26 0($25)
Rs = 25, Rt = 26

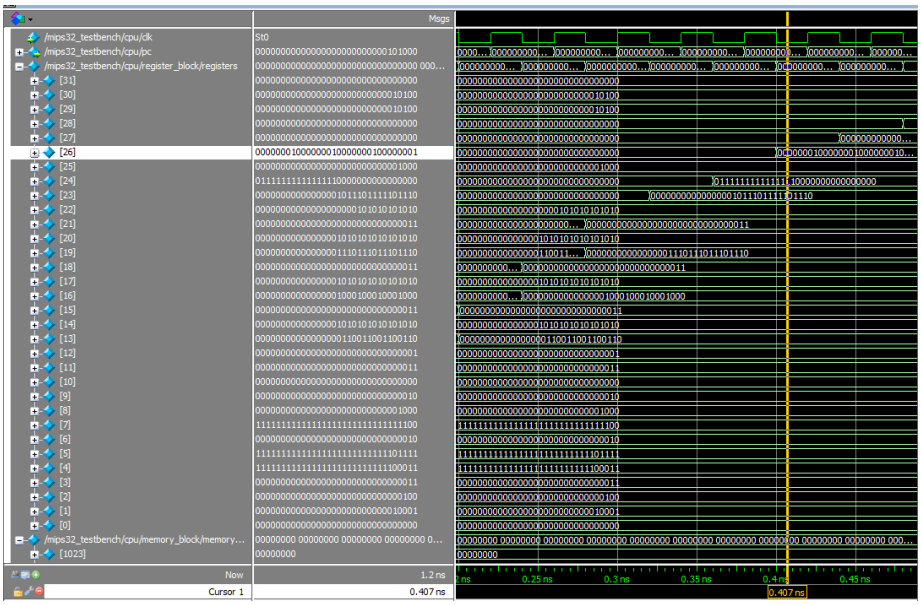
Expected Binary Values:

R[26] = 00000001000000010000000100000001,

R[25] = 00000000000000000000000000001000,

Immediate: 0000000000000000

PASS!



Before

Instruction: 100011 11001 11011 0000000000000100

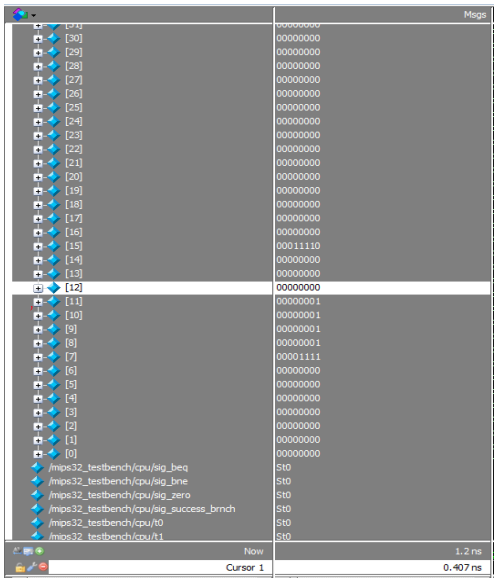
lw $27 4($25)
Rs = 25, Rt = 27

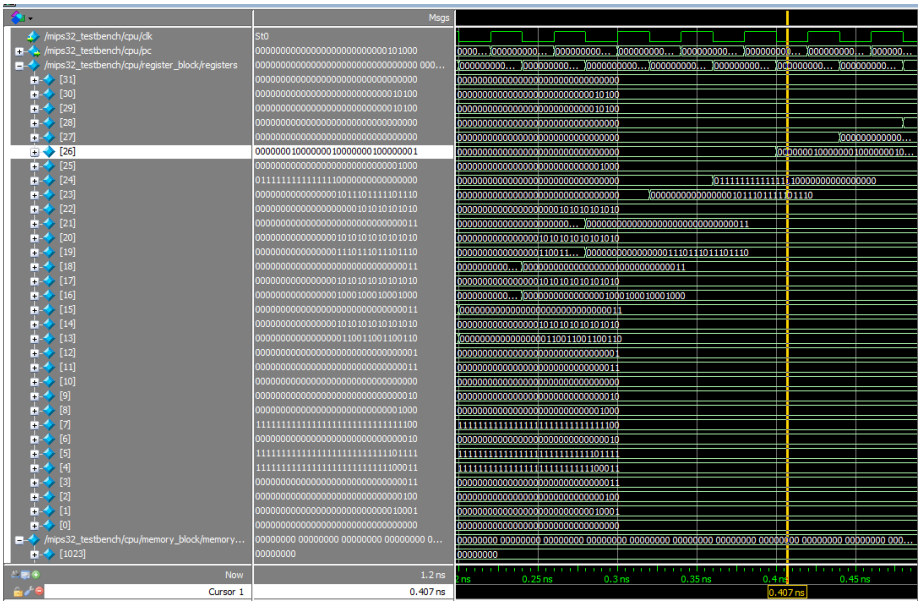
Initial Decimal Values:

R[27] = 0,

R[25] =8,

Immediate: 0000000000000100





After

Instruction: 100011 11001 11011 0000000000000100

lw $27 4($25)
Rs = 25, Rt = 27

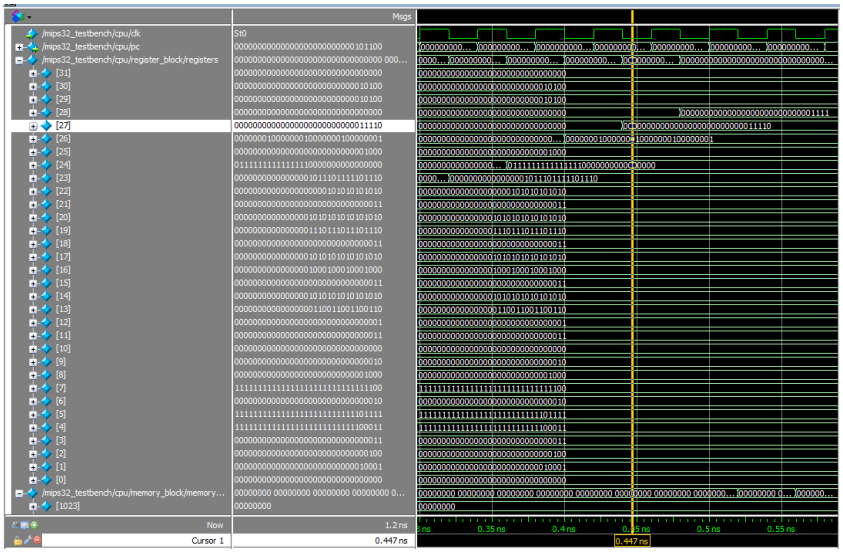
Expected Binary Values:

R[27] = 00000000000000000000000000011110,

R[25] =00000000000000000000000000001000,

Immediate: 0000000000000100

PASS!



Before

Instruction: 100011 11001 11100 1111111111111100

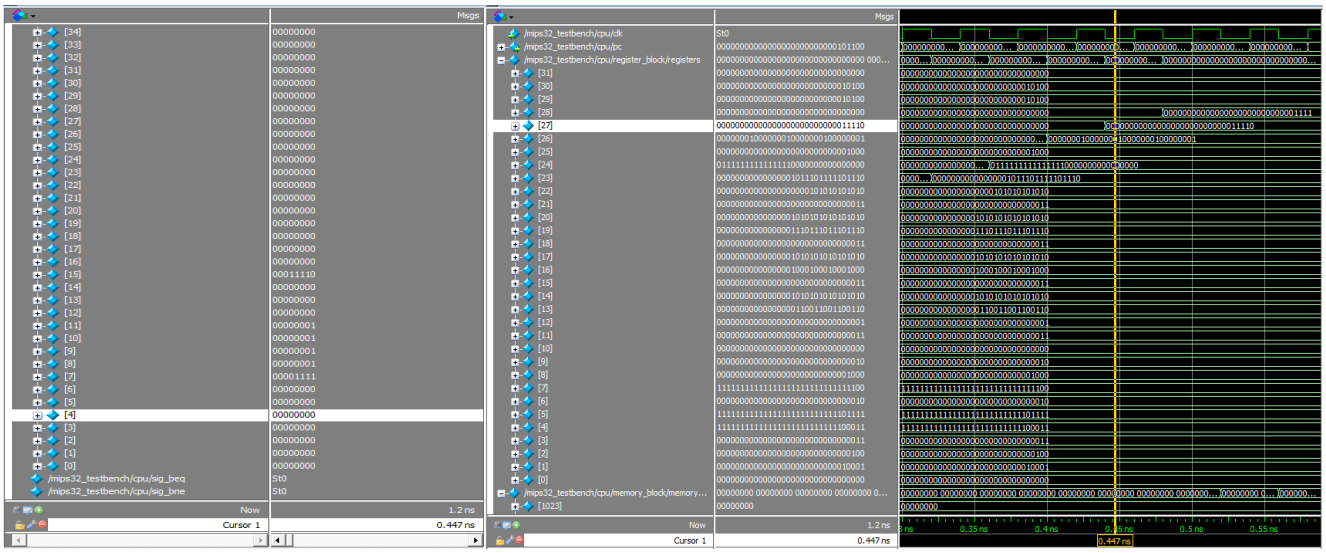
lw $28 -4($25)
Rs = 25, Rt = 28

Initial Decimal Values:

R[28] = 0,

R[25] =8,

Immediate: 1111111111111100



After

Instruction: 100011 11001 11100 1111111111111100

lw $28 -4($25)
Rs = 25, Rt = 28

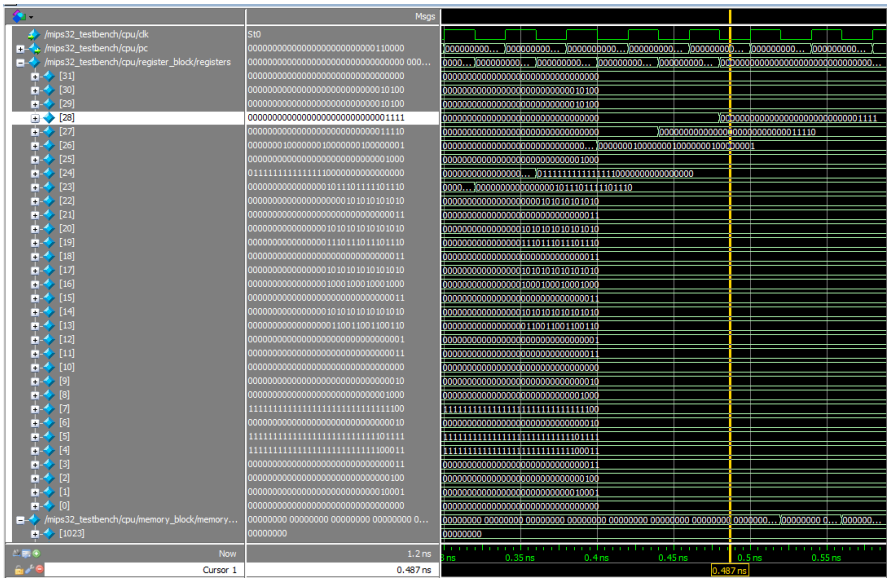
Expected Binary Values:

R[28] = 00000000000000000000000000001111,

R[25] =00000000000000000000000000001000,

Immediate: 1111111111111100

PASS!



Before

Instruction: 101011 11101 11001 0000000000000000

sw $25 0($29)
Rs = 29, Rt = 25

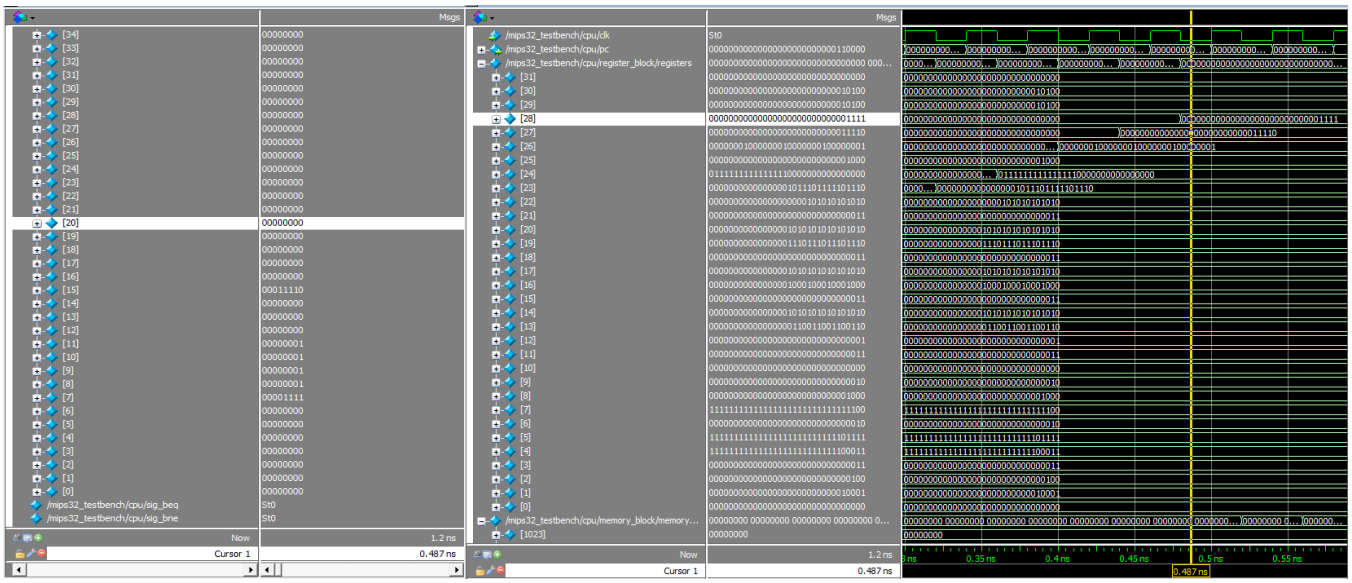
Initial Decimal Values:

M[20] = 0

R[29] = 20,

R[25] =8,

Immediate: 0000000000000000



After

Instruction: 101011 11101 11001 0000000000000000

sw $25 0($29)
Rs = 29, Rt = 25

Expected Decimal Values:

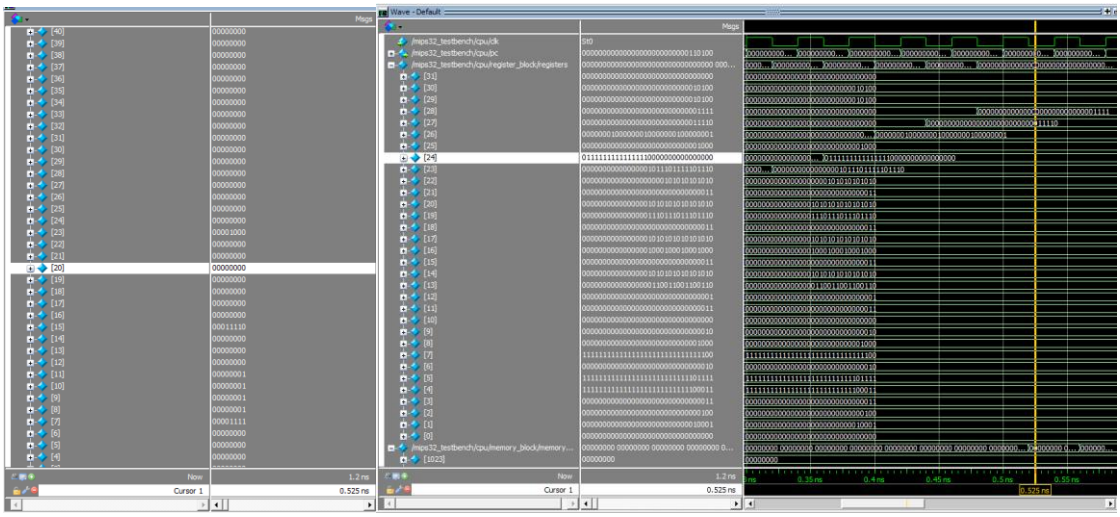
M[20] = 8,

R[29] = 20,

R[25] =8,

Immediate: 0000000000000000

PASS!



Before

Instruction: 101011 11101 11001 0000000000000100

sw $25 4($29)
Rs = 29, Rt = 25

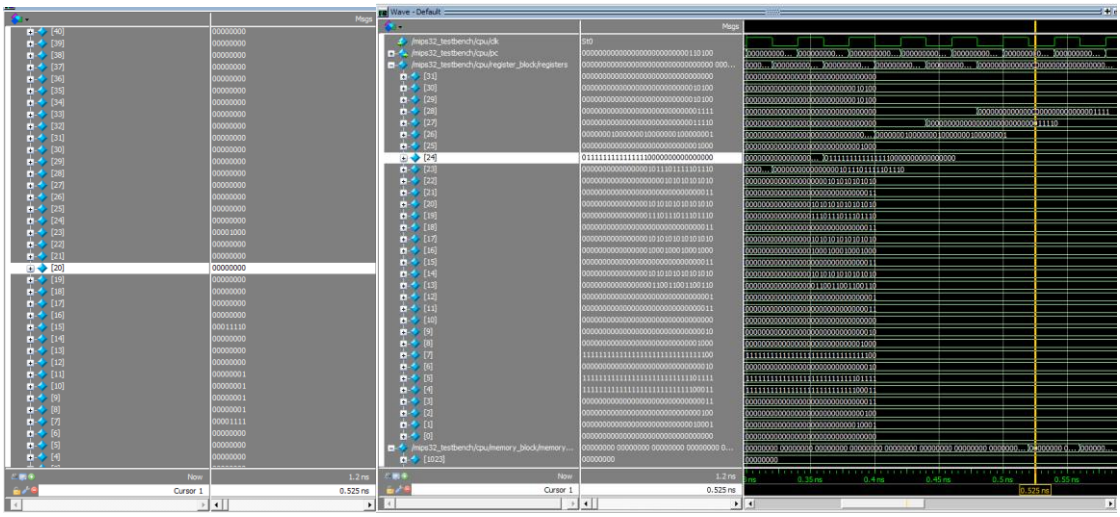
Initial Decimal Values:

M[24] = 0

R[29] = 20,

R[25] =8,

Immediate: 0000000000000100



After

Instruction: 101011 11101 11001 0000000000000100

sw $25 4($29)
Rs = 29, Rt = 25

Expected Decimal Values:

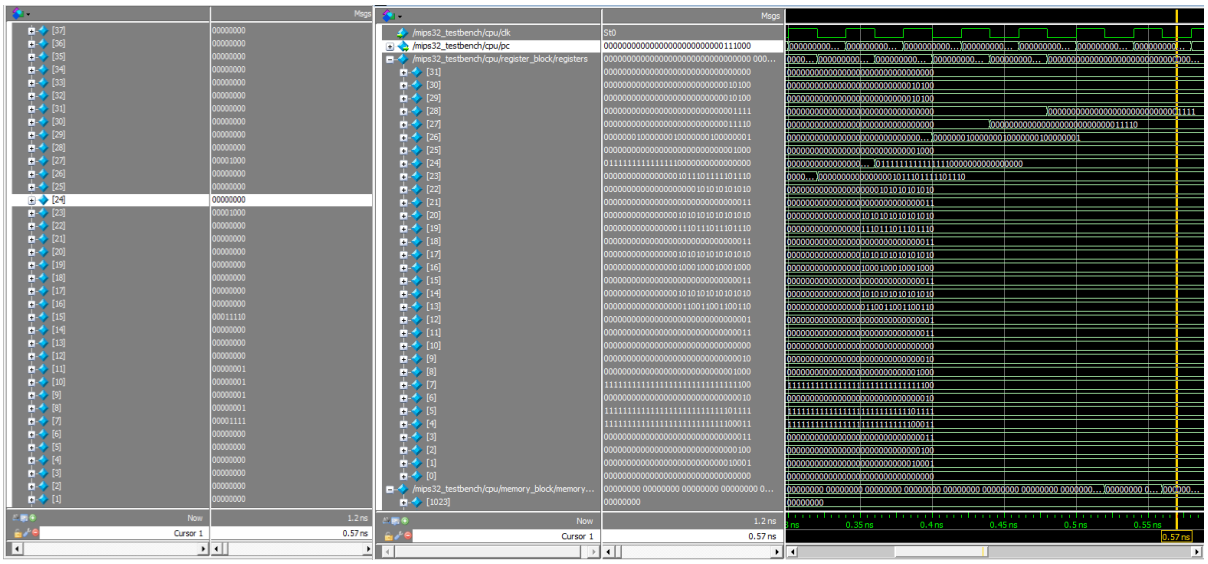
M[24] = 8,

R[29] = 20,

R[25] =8,

Immediate: 0000000000000100

PASS!



Before

Instruction: 101011 11101 11001 1111111111111100

sw $25 -4($29)
Rs = 29, Rt = 25

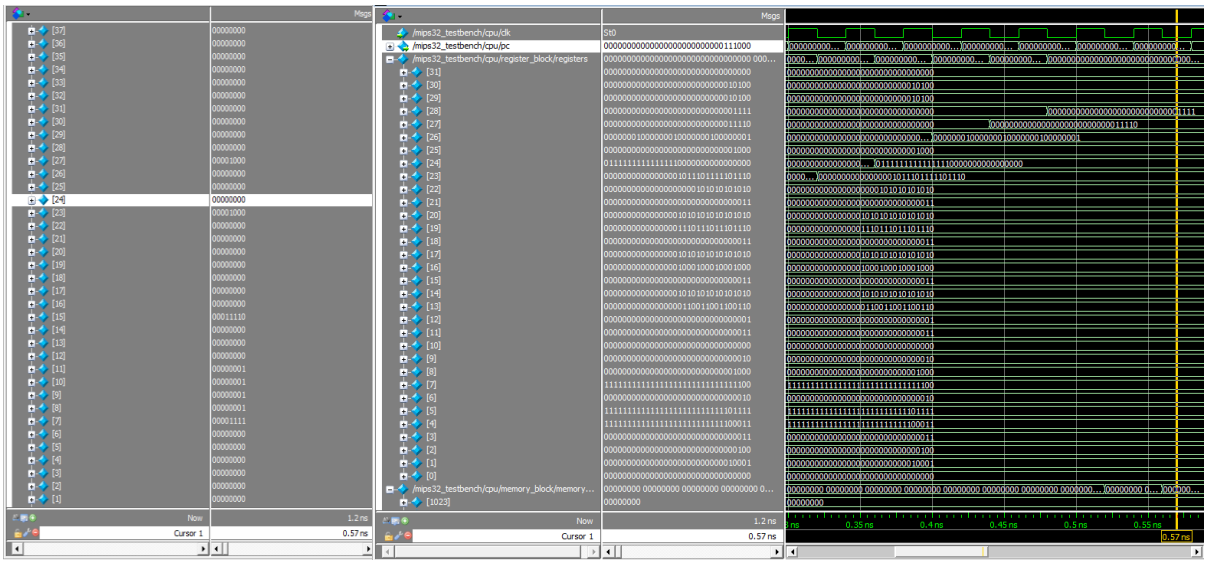
Initial Decimal Values:

M[16] = 0

R[29] = 20,

R[25] =8,

Immediate: 1111111111111100



After

Instruction: 101011 11101 11001 1111111111111100

sw $25 -4($29)
Rs = 29, Rt = 25

Expected Decimal Values:

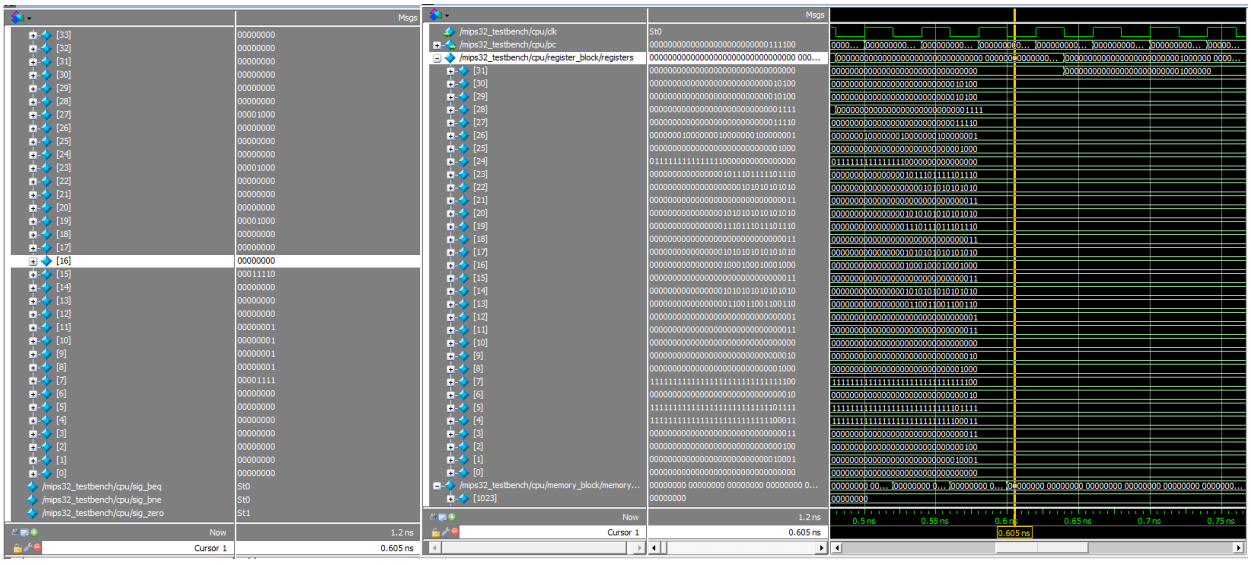
M[16] = 8,

R[29] = 20,

R[25] =8,

Immediate: 1111111111111100

PASS!



Before

Instruction: 000011 00000000000000000000010101

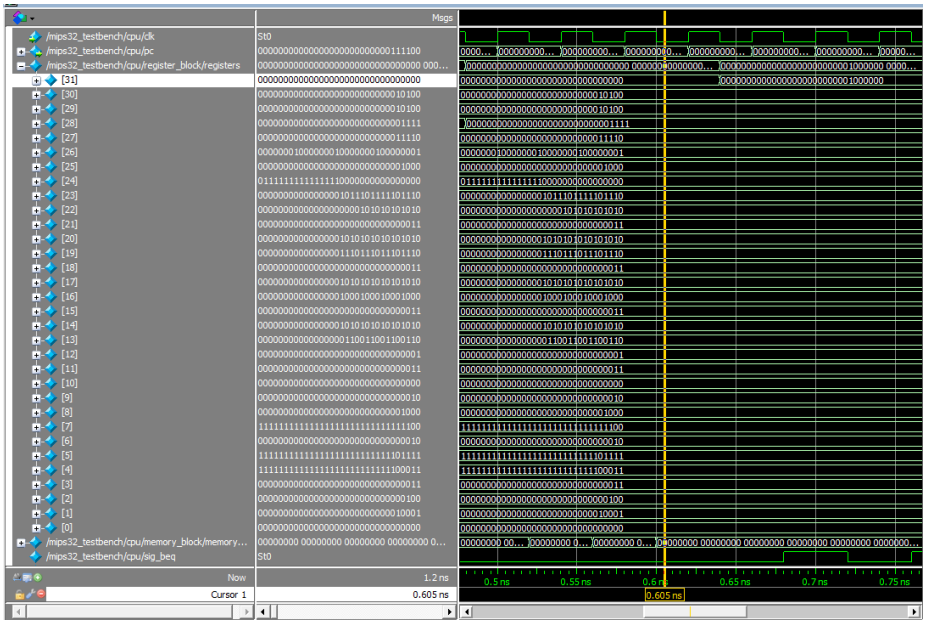
Jal 00000000000000000000010101

Initial Binary Values:

R[31]=32'b0

PC=00000000000000000000000111100

Jump Addr: 00000000000000000000001010100



After

Instruction: 000011 00000000000000000000010101

Jal 00000000000000000000010101

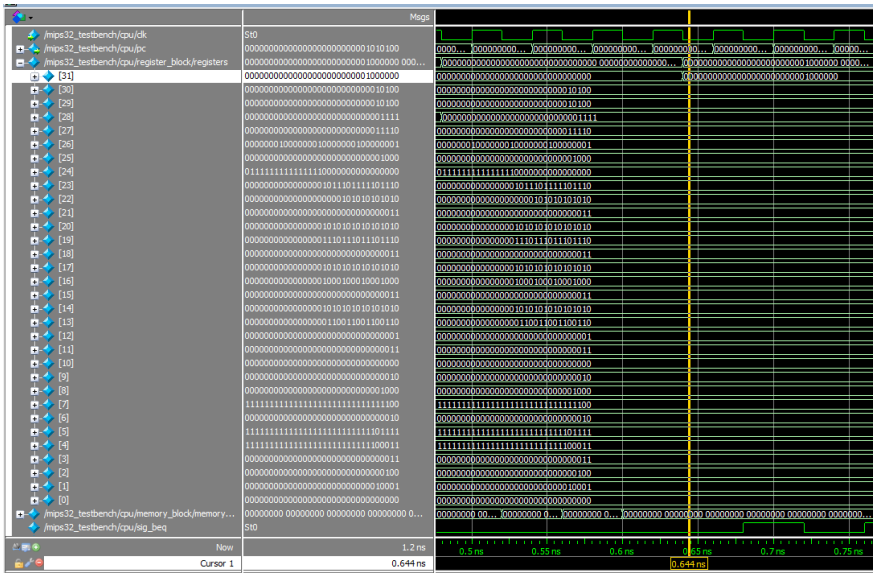
Initial Binary Values:

R[31]=PC+4=00000000000000000000000001000000

PC=00000000000000000000000001010100

Jump Addr: 00000000000000000000001010100

PASS!



Before

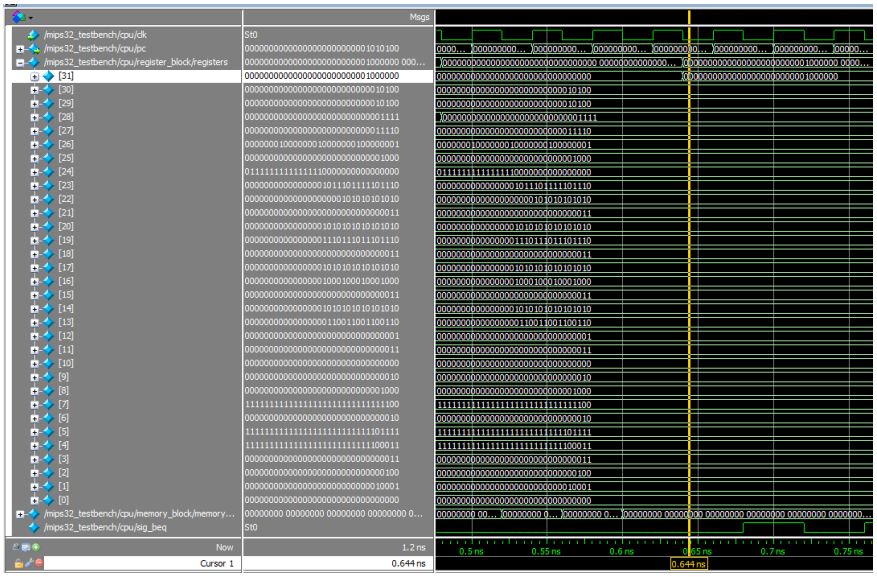
Instruction: 000000 11111 00000 00000 00000 001000

Jr $31

Initial Binary Values:

R[31]= 00000000000000000000000000001000000

PC=00000000000000000000000000001010100



After

Instruction: 000000 11111 00000 00000 00000 001000

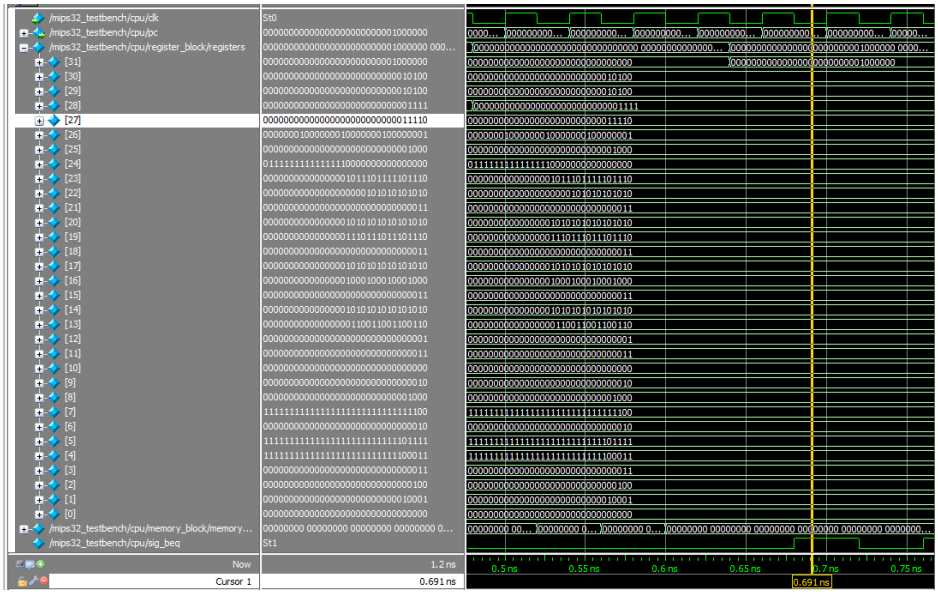
Jr $31

Expected Binary Values:

R[31]= 00000000000000000000000000001000000

PC=00000000000000000000000000001000000

PASS!



Before

Instruction: 000100 11101 11110 0000000000000011

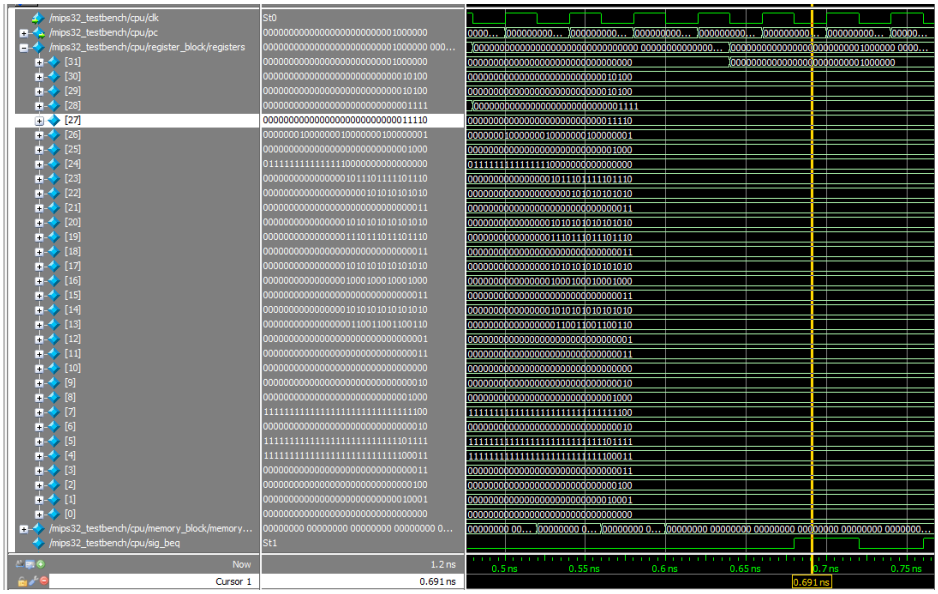
beq $29 $30 3

Initial Binary Values:

R[29]=8

R[30]=8

PC=00000000000000000000000001000000



After

Instruction: 000100 11101 11110 0000000000000011

beq $29 $30 3, Should branch to 3 instruction after PC+4 (+16 forward in binary)

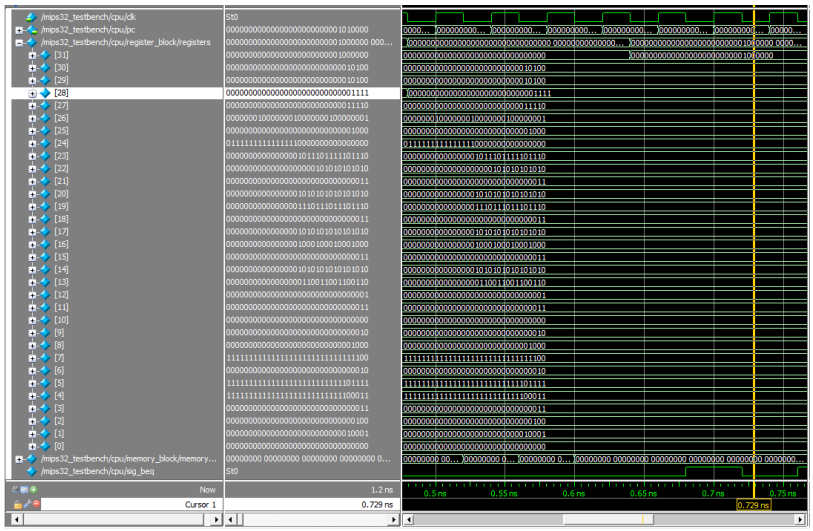
Expected Binary Values:

R[29]=8

R[30]=8

PC=00000000000000000000000001010000

PASS!



Before

Instruction: 000101 11101 00000 1111111111111100

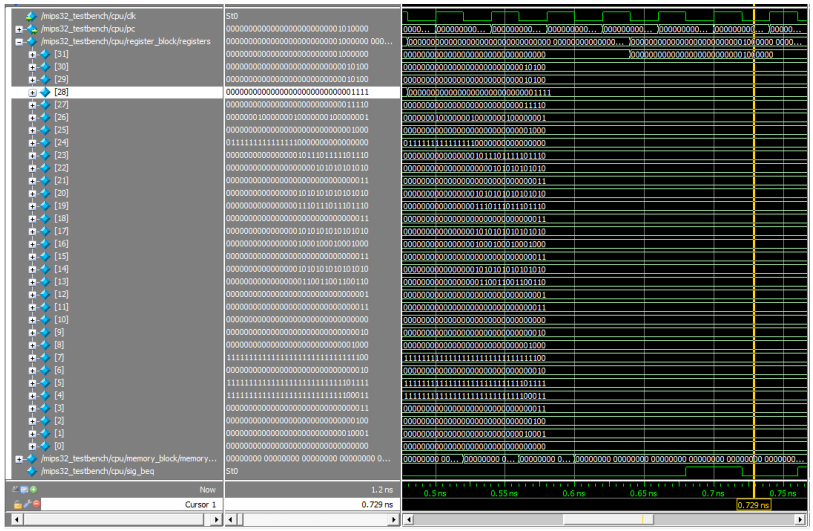
bne $29 $0 -4

Initial Binary Values:

R[0]=0

R[29]=8

PC=00000000000000000000000001010000



After

Instruction: 000101 11101 00000 1111111111111100

bne $29 $0 -4, Should branch to 4 instruction before PC+4 (-12 backward in binary)

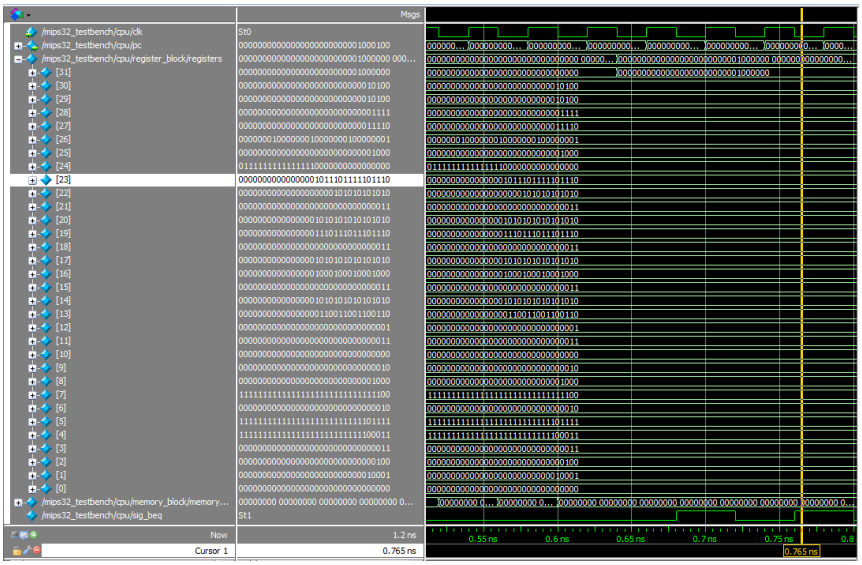
Expected Binary Values:

R[0]=0

R[29]=8

PC=00000000000000000000000001000100

PASS!



Before

Instruction: 000100 11101 00000 1111111111111110

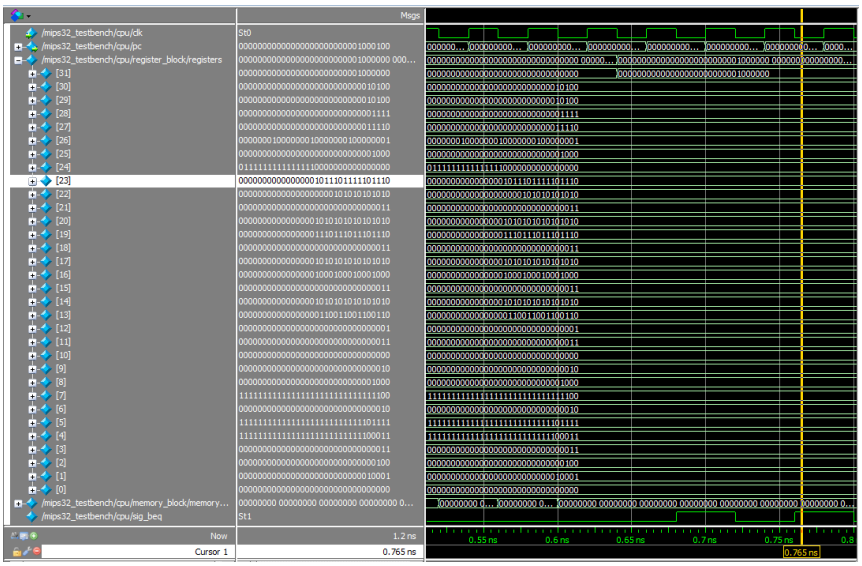
beq $29 $0 -1

Initial Binary Values:

R[0]=0

R[29]=8

PC=00000000000000000000000001000100



After

Instruction: 000100 11101 00000 1111111111111110

beq $29 $0 -1 (should fail)

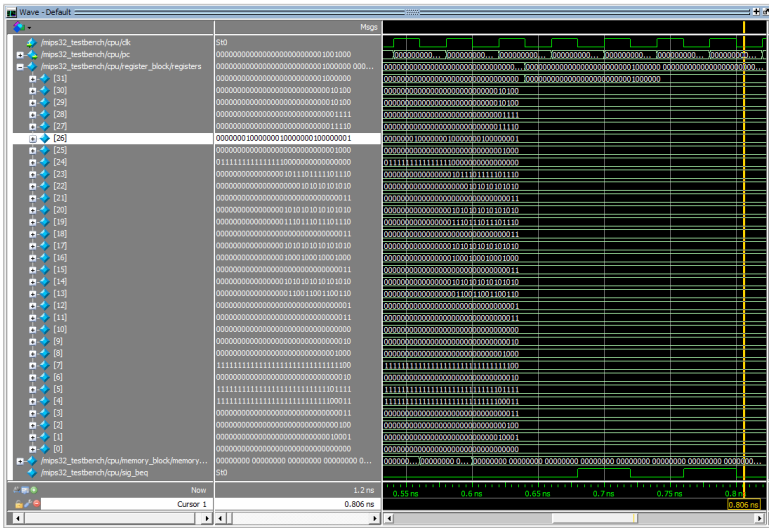
Initial Binary Values:

R[0]=0

R[29]=8

PC=00000000000000000000000001000100

PASS!



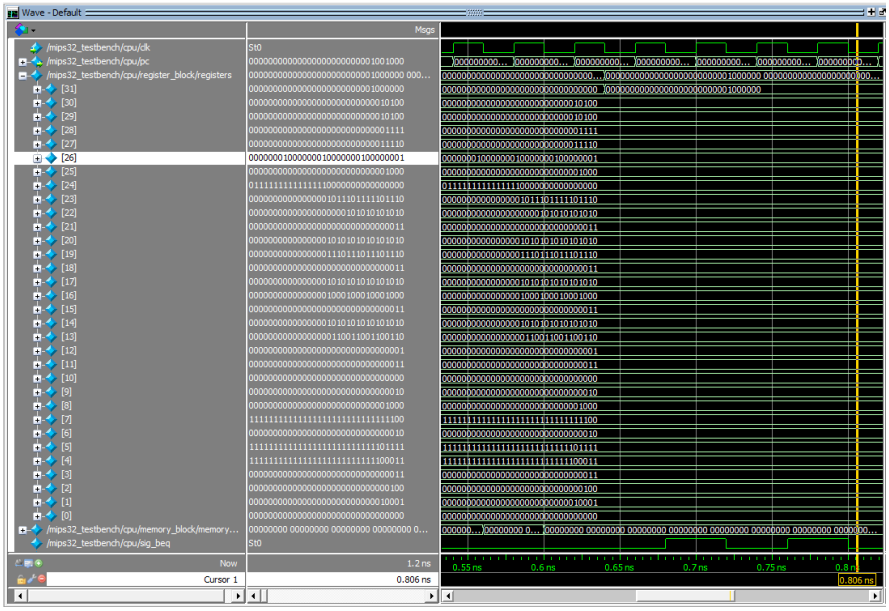
Before

Instruction: 000010 00000000000000000000010110

j 00000000000000000000000001011000

Initial

PC=00000000000000000000000001000100



After

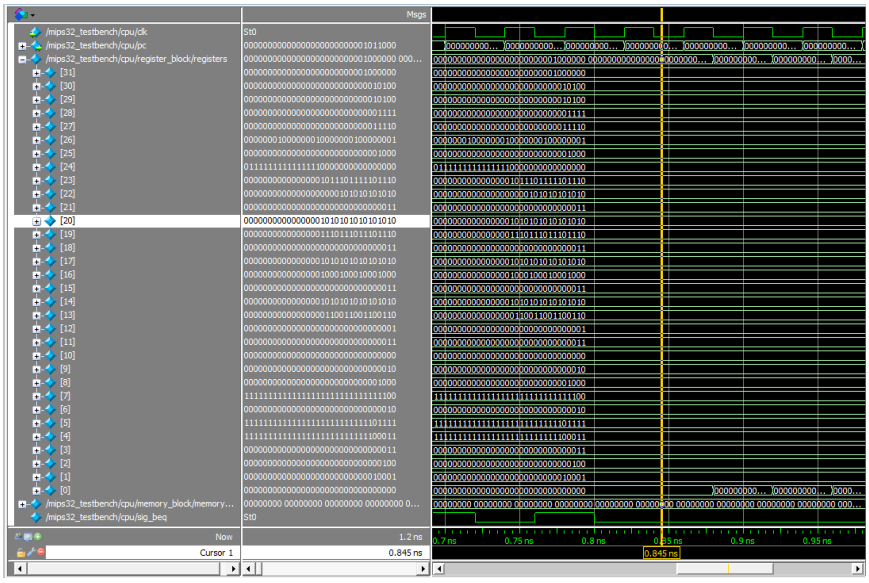
Instruction: 000010 00000000000000000000010110

j 00000000000000000000000001011000

Expected

PC=00000000000000000000000001011000

PASS!



DISCLAIMER: There are 21 instruction test here. There are not 28 but these are enough for all cases. For example there are 1 test case for logics(xor,and…) but their all edges are tested (0011, 0101). Some instructions even tested 3 times.

All In One

```
00000000001000100001100000100000//add $3 $1 $2
00000001000010100110000000100000//add $6 $4 $5
00000000111010000100100000100010//sub $9 $7 $8
00000001010010110110000000100010//sub $12 $10 $11
00000001101011100111100000100110//xor $15 $13 $14
00000010000100011001000000100100//and $18 $16 $17
00000010011101001010100000100101//or $21 $19 $20
00110110110101111011001111001100//ori $23 $22
00111100000110000111111111111111//lui $24
10001111001110100000000000000000//lw $26 $25
10001111001110110000000000000100//lw $27 $25
10001111001111001111111111111100//lw $28 $25
10101111101110010000000000000000//sw $25 $29
10101111101110010000000000000100//sw $25 $29
10101111101110011111111111111100//sw $25 $29
00001100000000000000000000010101//jal jump to jr
00010011101111100000000000000011//beq go 3 after pc+4 $29 == $30
00010011101000001111111111111110//beq fail $29 == $0
00001000000000000000000000010110//j jump to end
00000000000000000000000000000000//to show this is not worked, showing it is jumped
00010111101000001111111111111100//bne go back 4 after pc + 4
00000011111000000000000000001000//jr jump to beq $31
00000000000000000000000000000000
```

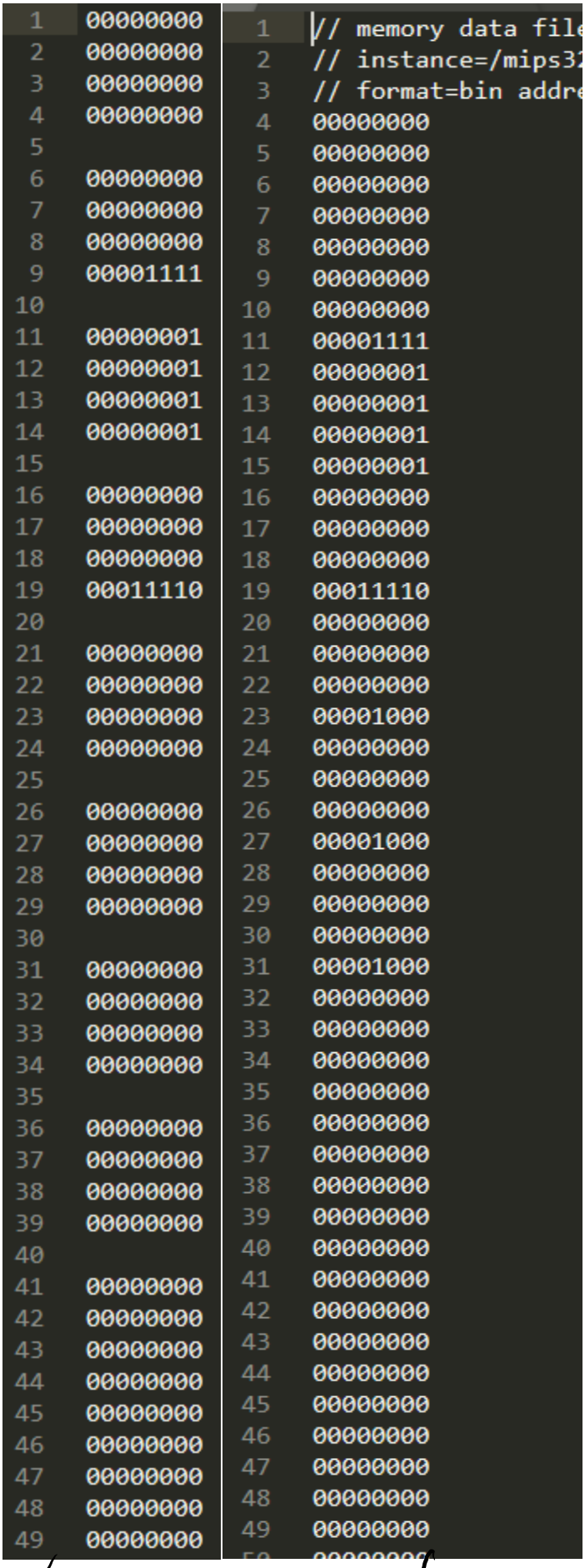
instruction.mem

```
00000000000000000000000000000000
00000000000000000000000000001101//1==13
00000000000000000000000000000100//2==4
00000000000000000000000000000000//3
11111111111111111111111111110100//4==-12
11111111111111111111111111101111//5==-17
00000000000000000000000000000000//6
00000000000000000000000000000100//7==4
00000000000000000000000000001000//8==8
00000000000000000000000000000000//9
00000000000000000000000000000011//10==3
00000000000000000000000000000011//11==3
00000000000000000000000000000000//12
00000000000000001100110011001100//13
00000000000000001010101010101010//14
00000000000000000000000000000000//15
00000000000000001100110011001100//16
00000000000000001010101010101010//17
00000000000000000000000000000000//18
00000000000000001100110011001100//19
00000000000000001010101010101010//20
00000000000000000000000000000000//21
00000000000000001010101010101010//22
00000000000000000000000000000000//23
00000000000000000000000000000000//24
00000000000000000000000000001000//25==3. mem blk
00000000000000000000000000000000//26
00000000000000000000000000000000//27
00000000000000000000000000000000//28
00000000000000000000000000010100//29
00000000000000000000000000010100//30
00000000000000000000000000000000//31
```

registers.mem

```
// memory data file (do not edit t
// instance=/mips32_testbench/cpu/
// format=bin addressradix=h datar
00000000000000000000000000000000
00000000000000000000000000010001
00000000000000000000000000000100
00000000000000000000000000000011
11111111111111111111111111100011
11111111111111111111111111101111
00000000000000000000000000000010
11111111111111111111111111111100
00000000000000000000000000001000
00000000000000000000000000000010
00000000000000000000000000000000
00000000000000000000000000000011
00000000000000000000000000000001
00000000000000001100110011001100
00000000000000001010101010101010
00000000000000000000000000000011
00000000000000001000100010001000
00000000000000001010101010101010
00000000000000000000000000000011
00000000000000001110111011101110
00000000000000001010101010101010
00000000000000000000000000000011
00000000000000000010101010101010
00000000000000001011101111101110
01111111111111111000000000000000
00000000000000000000000000001000
00000001000000010000000100000001
00000000000000000000000000011110
00000000000000000000000000001111
00000000000000000000000000010100
00000000000000000000000000010100
00000000000000000000000001000000
```

registers_out.mem



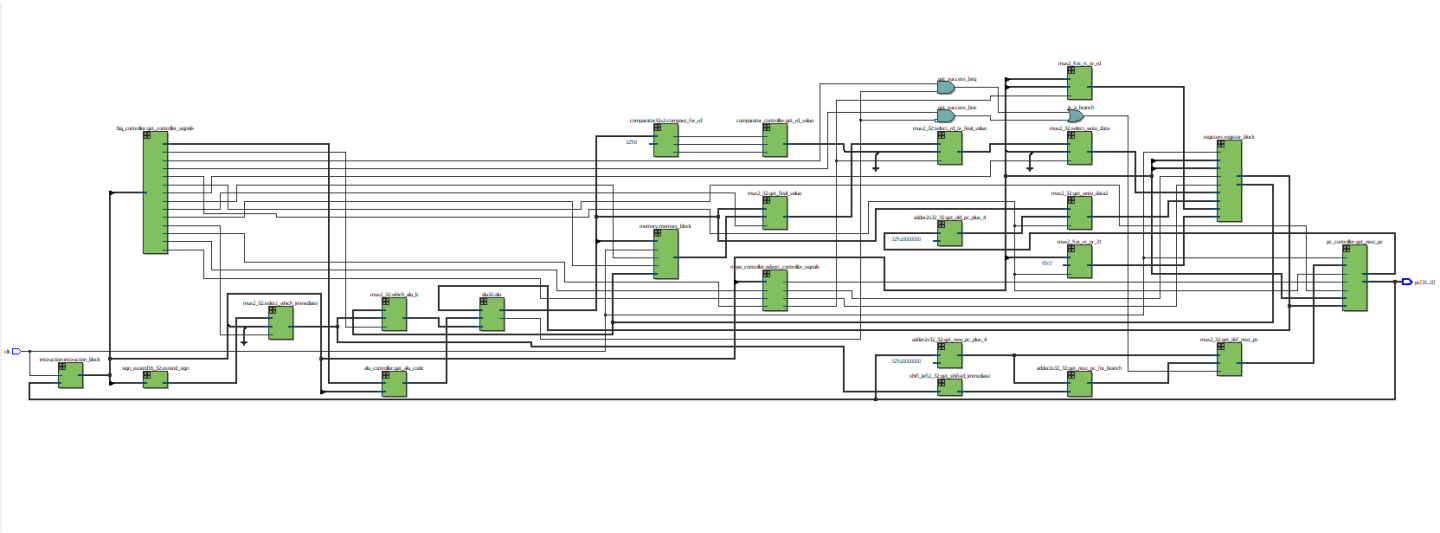
```
00000000
00000000
00000000
00000000

00000000
00000000
00000000
00001111

00000001
00000001
00000001
00000001

00000000
00000000
00000000
00011110

00000000
00000000
00000000
00000000

00000000
00000000
00000000
00000000

00000000
00000000
00000000
00000000

00000000
00000000
00000000
00000000

00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

```
// memory data file
// instance=/mips32
// format=bin addre
00000000
00000000
00000000
00000000
00000000
00000000
00001111
00000001
00000001
00000001
00000001
00000000
00000000
00000000
00011110
00000000
00000000
00000000
00001000
00000000
00000000
00000000
00001000
00000000
00000000
00000000
00001000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

data.mem

data_out.mem

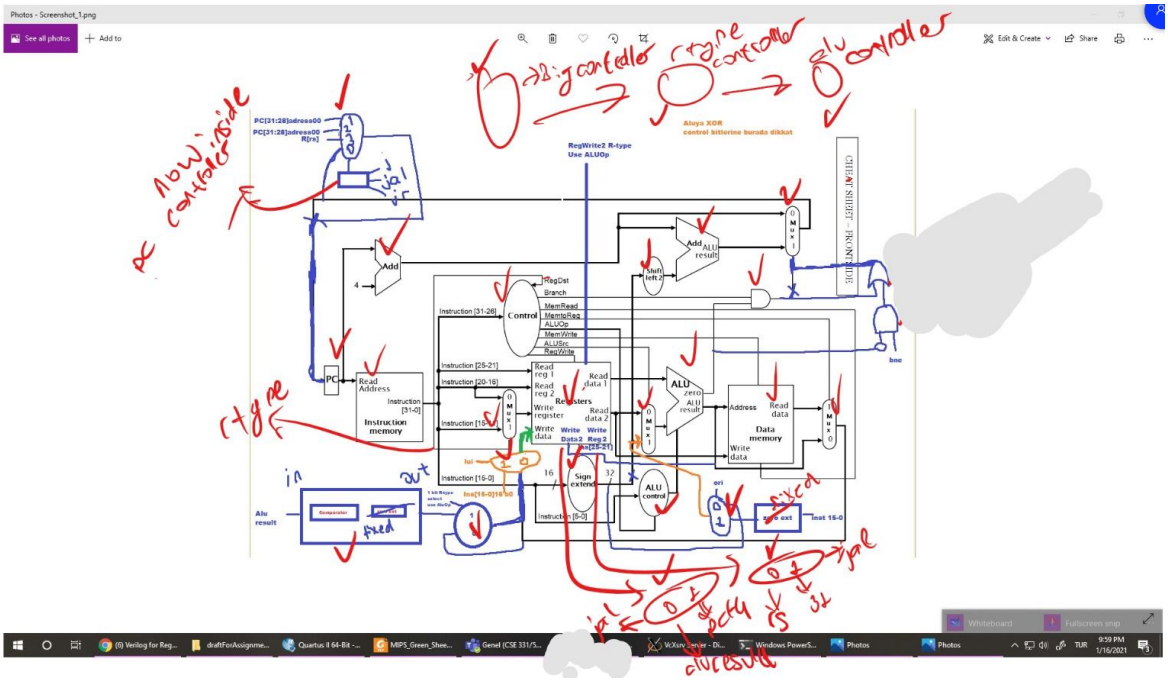
RTL View



PLAYGROUND

These are my things that I did while doing this homework.

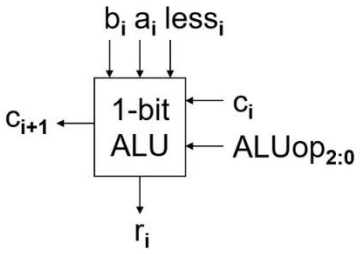
DISCLAIMER: These may not represent the final form as I do changes on the fly and doesn't reflect them in these.



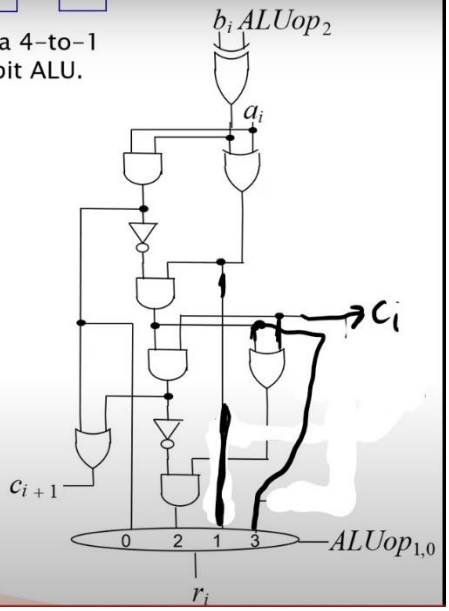
cse 331

1-Bit ALU

- The full adder, an xor gate, and a 4-to-1 mux are combined to form a 1-bit ALU.



ALUOp	Function
000	AND
001	OR
010	ADD
110	SUBTRACT
011	XOR



Comparator
two's complement fix

+ = 0
− = 1

a_+	b_+	lt	eq	gt
+	+	1	x	x
x	x	x	1	x
+	+	x	x	1
−	+	1	x	x
−	−	1	x	x
+	−	1	x	x
−	+	x	x	1
−	−	x	x	1
+	−	x	x	1

lt_0	eq_0	gt_0
1	0	0
0	1	0
0	0	1
1	0	0
0	0	1
0	0	1
1	0	0
1	0	0
0	0	1

$lt_0 = lt_{sb'} + gt_{sa}$

$gt_0 = gt_{sb'} + lt_{sb}$

$eq_0 = eq$

*** Big Control**

	opcodes	00 0000	00 0010	00 0011	10 0011	10 1011	00 0100	00 0101	00 1101	00 1111
		R-Type	J	JAL	LW	SW	BEQ	BNE	ORI	LUI
		<c>	<c>	<c>	<c>	<c>	<c>	<c>	<c>	<c>
R-Type	RegDst/Rtype	1	0	0	0	0	0	0	0	0
LW v SW v ORI	ALUSrc	0	0	0	1	1	0	0	1	0
LW	MemToReg	0	0	0	1	0	0	0	0	0
R-Type v LW v ORI v LUI	RegWrite	1	0	0	1	0	0	0	1	1
R-Type v JAL	RegWrite2	1	0	1	0	0	0	0	0	0
LW	MemRead	0	0	0	1	0	0	0	0	0
SW	MemWrite	0	0	0	0	1	0	0	0	0
J	J	0	1	0	0	0	0	0	0	0
JAL	JAL	0	0	1	0	0	0	0	0	0
BEQ	BEQ	0	0	0	0	0	1	0	0	0
BNE	BNE	0	0	0	0	0	0	1	0	0
ORI	ORi	0	0	0	0	0	0	0	1	0
LUI	LUi	0	0	0	0	0	0	0	0	1
	ALUOp(Symbolic)	R-Type	X	X	Add	Add	Sub	Sub	OR	X
R-Type v ORI	ALUOp_{1}	1	0	0	0	0	0	0	1	0
R-Type v BEQ v BNE	ALUOp_{0}	1	0	0	0	0	1	1	0	0

* RType Controller

- Will create JR_{i} signal.
- Reforms RType because of JR being R type.

In			In	Out			Out
Function			00 1000				
RegDst/Rtype	RegWrite	RegWrite2	JR_{i}	RType	RegWrite	RegWrite2	JR_{o}
<c>	<c>	<c>	<c>	<c>	<c>	<c>	<c>
1	1	1	1	0	0	0	1
1	1	1	0	1	1	1	0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	0

RType := RegDst/Rtype * !JR_{i}
JR_{o}:= RegDst/Rtype * JR_{i}
RegWrite := RegWrite * !JR_{i}
RegWrite2 := RegWrite2 * !JR_{i}

* PC Controller

- Will take J and JAL from Big Controller, JR from ALUOp Controller.

J	JAL	JR	PCMux_{1}	PCMux_{0}
<c>	<c>	<c>	<c>	<c>
0	0	0	0	0
0	0	1	1	0
1	0	0	0	1
0	1	0	0	1

PCMux_{1} := !J * !JAL * JR
PCMux_{0} := J * !JAL * !JR + !J * JAL * !JR = !JR(J or JAL)

*** ALU Control Unit**

*** ALU Controller**

ALUOp_{1}	ALUOp_{0}	F5_{}	F4_{}	F3_{}	F2_{}	F1_{}	F0_{}	OP2	OP1	OP0
0	0	X	X	X	X	X	X	0	1	0
0	1	X	X	X	X	X	X	1	1	0
1	0	X	X	X	X	X	X	0	0	1
1	1	1	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	1	0	1	0
1	1	1	0	0	0	1	1	0	1	1
1	1	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	1	0	0	1

```
OP2 := !ALUOp1 * ALUOp0 + ALUOp1 * ALUOp0 * !F2 * F1 * !F0
OP1 := !ALUOp1 + ((!F2 * !F0) + (F1 * !F0))*ALUOp1*ALUOp0
OP0 := ALUOp1 * !ALUOp0 + ALUOP1*ALUOp0*F2*(*F1* xor F0)
```

*** Comparator Controller**

lt	eq	gt	o_{1}	o_{0}
<c>	<c>	<c>	<c>	<c>
1	0	0	1	0
0	1	0	0	1
0	0	1	1	1

```
o_{1} := lt xor gt
o_{0} := eq xor gt
```

Testbenches

There are also many testbenches for other smaller parts of my design.

```
VSIM 30> step -current
# time =  0, a =00011000000000000000010001000001, b=00011000011001000010001000000001, out=00110000011001000010011001000010, expected=00110000011001000010011001000010, tresult=1
# time = 20, a =00011000000000000000010001001101, b=00011000011001000010001000000101, out=00110000011001000010011001010010, expected=00110000011001000010011001010010, tresult=1

VSIM 31>
```

adder2x32_32

```
VSIM 39> step -current
# time =  0, a =1, b=0, ci=0,  alu_code=000, ri=0, exp_ri=0, cipl=0, exp_cipl=0, tresult=1
# time = 20, a =1, b=1, ci=0,  alu_code=000, ri=1, exp_ri=1, cipl=1, exp_cipl=1, tresult=1
# time = 40, a =1, b=0, ci=0,  alu_code=001, ri=1, exp_ri=1, cipl=0, exp_cipl=0, tresult=1
# time = 60, a =1, b=1, ci=0,  alu_code=001, ri=1, exp_ri=1, cipl=1, exp_cipl=1, tresult=1
# time = 80, a =0, b=0, ci=0,  alu_code=001, ri=0, exp_ri=0, cipl=0, exp_cipl=0, tresult=1
# time = 100, a =0, b=0, ci=0,  alu_code=010, ri=0, exp_ri=0, cipl=0, exp_cipl=0, tresult=1
# time = 120, a =0, b=1, ci=0,  alu_code=010, ri=1, exp_ri=1, cipl=0, exp_cipl=0, tresult=1
# time = 140, a =1, b=1, ci=0,  alu_code=010, ri=0, exp_ri=0, cipl=1, exp_cipl=1, tresult=1
# time = 160, a =1, b=1, ci=1,  alu_code=010, ri=1, exp_ri=1, cipl=1, exp_cipl=1, tresult=1
# time = 180, a =1, b=1, ci=1,  alu_code=110, ri=0, exp_ri=0, cipl=1, exp_cipl=1, tresult=1
# time = 200, a =1, b=0, ci=1,  alu_code=110, ri=1, exp_ri=1, cipl=1, exp_cipl=1, tresult=1
# time = 220, a =0, b=0, ci=1,  alu_code=110, ri=0, exp_ri=0, cipl=1, exp_cipl=1, tresult=1
# time = 240, a =1, b=1, ci=0,  alu_code=011, ri=0, exp_ri=0, cipl=1, exp_cipl=1, tresult=1
# time = 260, a =1, b=0, ci=0,  alu_code=011, ri=1, exp_ri=1, cipl=0, exp_cipl=0, tresult=1
# time = 280, a =0, b=1, ci=0,  alu_code=011, ri=1, exp_ri=1, cipl=0, exp_cipl=0, tresult=1
# time = 300, a =0, b=0, ci=0,  alu_code=011, ri=0, exp_ri=0, cipl=0, exp_cipl=0, tresult=1

VSIM 40>
```

Alu1

```
# time =  0, a =00011000000000000000010001000001, b=00011000011001000010001000000001, alu_result=00011000000000000000000000000001, exp_alu_result=00011000000000000000000000000001, alu_code=000, tresult=1
# time = 20, a =00011000000000000000010001000001, b=00011000011001000010001000000001, alu_result=00011000011001000010011001000001, exp_alu_result=00011000011001000010011001000001, alu_code=001, tresult=1
# time = 40, a =00011000000000000000010001000001, b=00011000011001000010001000000001, alu_result=00110000011001000010011001000010, exp_alu_result=00110000011001000010011001000010, alu_code=010, tresult=1
# time = 60, a =00000000000000011110001001000000, b=00000000000001010001101000001001, alu_result=11111111111111110011001000000110111, exp_alu_result=11111111111111110011001000000110111, alu_code=110, tresult=1
# time = 80, a =00000000000001010001101000001001, b=00000000000000011110001001000000, alu_result=00000000000000011001101111001001, exp_alu_result=00000000000000011001101111001001, alu_code=110, tresult=1
# time = 100, a =00111000000000000000010001000001, b=00011000011001000010001000000001, alu_result=00100000011001000010011001000000, exp_alu_result=00100000011001000010011001000000, alu_code=011, tresult=1
```

Alu32

```
VSIM 47> step -current
# time =  0, alu_op=002, fnctn=011001, alu_code=010, expected= 010, tresult=1
# time = 20, alu_op=002, fnctn=011111, alu_code=010, expected= 010, tresult=1
# time = 40, alu_op=012, fnctn=011111, alu_code=110, expected= 110, tresult=1
# time = 60, alu_op=102, fnctn=011111, alu_code=001, expected= 001, tresult=1
# time = 80, alu_op=102, fnctn=010101, alu_code=001, expected= 001, tresult=1
# time = 100, alu_op=112, fnctn=100000, alu_code=010, expected= 010, tresult=1
# time = 120, alu_op=112, fnctn=100010, alu_code=110, expected= 110, tresult=1
# time = 140, alu_op=112, fnctn=100110, alu_code=011, expected= 011, tresult=1
# time = 160, alu_op=112, fnctn=100100, alu_code=000, expected= 000, tresult=1
# time = 180, alu_op=112, fnctn=100101, alu_code=001, expected= 001, tresult=1

VSIM 48>
```

Alu_controller

```
VSIM 49> step -current
# time =  0, a =0, b=1, c=1, out=0, expected=0, tresult=1
# time = 20, a =1, b=1, c=1, out=1, expected=1, tresult=1
# time = 40, a =0, b=0, c=0, out=0, expected=0, tresult=1

VSIM 50>
```

And1x3_1

```
VSIM 51> step -current
# time =  0, a =0, b=1, c=1, d=1, out=0, expected=0, tresult=1
# time = 20, a =1, b=1, c=1, d=1, out=1, expected=1, tresult=1
# time = 40, a =0, b=0, c=0, d=0, out=0, expected=0, tresult=1

VSIM 52>
```

And1x4_1

```
VSIM 53> step -current
# time =  0, a =011111, out=0, expected=0, tresult=1
# time = 20, a =111111, out=1, expected=1, tresult=1

VSIM 54>
```

And6x1_1

```
VSIM 57> step -current
# time =  0, r_type=1, expected_r_type=1, ,alu_src=0, expected_alu_src=0,mem_to_reg=0, expected_mem_to_reg=0,reg_write=1, expected_reg_write=1,reg_write2=1, expected_reg_write2=1,mem_read=0, expected_mem_read=0,mem_write=0, expected_mem_write=0,j=0, expected_j=0,jal=0,
expected_jal=0,beq=0, expected_beq=0,bne=0, expected_bne=0,ori=0, expected_ori=0,lui=0, expected_lui=0,alu_op=11, expected_alu_op=11

VSIM 58>
```

Big_controller

```
VSIM 59> step -current
# time =  0, a =0, b=0, lti=0, gti=0,  lt=0, eq=1, gt=0, tresult=1
# time = 20, a =1, b=0, lti=0, gti=0,  lt=0, eq=0, gt=1, tresult=1
# time = 40, a =0, b=1, lti=0, gti=0,  lt=1, eq=0, gt=0, tresult=1
# time = 60, a =1, b=1, lti=0, gti=0,  lt=0, eq=1, gt=0, tresult=1
# time = 80, a =1, b=1, lti=0, gti=1,  lt=0, eq=0, gt=1, tresult=1
# time = 100, a =1, b=1, lti=1, gti=0,  lt=1, eq=0, gt=0, tresult=1
# time = 120, a =1, b=0, lti=1, gti=0,  lt=1, eq=0, gt=0, tresult=1

VSIM 60>
```

Comparator1x2

```
VSIM 61> step -current
# time =  0, a =00000000000000000000000000000000, b=00000000000000000000000000000000, lt=0, eq=1, gt=0, tresult=1
# time = 20, a =00000000000000000001100000000000, b=00000000000000000000011000000000, lt=0, eq=0, gt=1, tresult=1
# time = 40, a =00000000000000000001100000000000, b=00000000000000001000110000000000, lt=1, eq=0, gt=0, tresult=1
# time = 60, a =11111111111111111011000011110100, b=00000000000000001000110000000000, lt=1, eq=0, gt=0, tresult=1
# time = 80, a =00000000000000000001111000001100, b=11111111111111111001110100000000, lt=0, eq=0, gt=1, tresult=1
# time = 100, a =00000000000000000001111000001100, b=00000000000000000001111000001100, lt=0, eq=1, gt=0, tresult=1

VSIM 62>
```

Comparator32x2

```
VSIM 63> step -current
# time =  0, lt=1, eq=0, gt=0, out=10, expected=10, tresult=1
# time = 20, lt=0, eq=1, gt=0, out=01, expected=01, tresult=1
# time = 40, lt=0, eq=0, gt=1, out=11, expected=11, tresult=1

VSIM 64>
```

Comparator_controller

```
VSIM 65> step -current
# time =  0, a =011111, b=011111, out=1, expected=1, tresult=1
# time = 20, a =011111, b=001111, out=0, expected=0, tresult=1

VSIM 66>
```

Equal_6x2

```
VSIM 67> step -current
# time =  0, a =1, b=0, select=0, out=1, expected=1, tresult=1
# time = 20, a =0, b=1, select=1, out=1, expected=1, tresult=1

VSIM 68>
```

Mux2_1

```
VSIM 69> step -current
# time =  0, a =1, b=0, c=0, d=0,  select=00, out=1, expected=1, tresult=1
# time = 20, a =0, b=1, c=0, d=0,  select=01, out=1, expected=1, tresult=1
# time = 40, a =0, b=0, c=1, d=0,  select=10, out=1, expected=1, tresult=1
# time = 60, a =0, b=0, c=0, d=1,  select=11, out=1, expected=1, tresult=1

VSIM 70>
```

Mux4_1

```
VSIM 71> step -current
# time =  0, out=0, expected=0, tresult=1
# time = 20, out=1, expected=1, tresult=1

VSIM 72>
```

Or1x32_1

```
VSIM 73> step -current
# time =  0, a =0, b=0, c=1, out=1, expected=1, tresult=1
# time = 20, a =0, b=0, c=0, out=0, expected=0, tresult=1

VSIM 74>
```

Or1x3_1

```
VSIM 75> step -current
# time =  0, a =0, b=0, c=1, d=0, out=1, expected=1, tresult=1
# time = 20, a =0, b=0, c=0, d=0, out=0, expected=0, tresult=1

VSIM 76>
```

Or1x4_1

```
VSIM 78> step -current
# time =  0, rtype_i=1, rtype=0, exp_rtype=0, fnctn=1000, jr=1, exp_jr=1, tresult=1
# time = 20, rtype_i=0, rtype=0, exp_rtype=0, fnctn=1000, jr=0, exp_jr=0, tresult=1
# time = 40, rtype_i=1, rtype=1, exp_rtype=1, fnctn=101000, jr=0, exp_jr=0, tresult=1
# time = 60, rtype_i=0, rtype=0, exp_rtype=0, fnctn=101000, jr=0, exp_jr=0, tresult=1

VSIM 79>
```

Rtype_controller

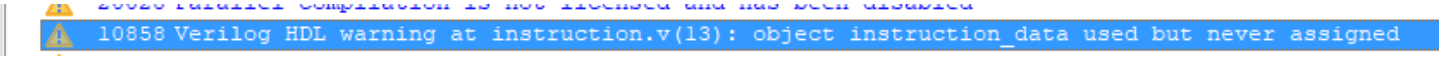
```
VSIM 80> step -current
# time =  0, out=11010111010001000111010111010001000, a=00111010111010001000111010110100010
# time = 20, out=00010111010001000111010111010001000, a=00000101110100010001110101101000010

VSIM 81>
```

Shift_left2_32

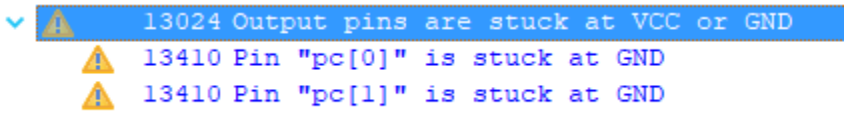
```
VSIM 82> step -current
# time =  0, out=00000000000000000011010110100010, a=0011010110100010
# time = 20, out=11111111111111111111010110100010, a=1011010110100010

VSIM 83>
```

Sign_extend16_32

Warnings Explained



Because initialized in mips32_testbench.v



This is expected as PC consists of bytes and we start from 0 and update by multiples of 4.

DISCLAIMER: PDF specifies that there will be no input to mips32 but there should be at least one because synthesizer errors about top level has no logic. Also there should be one output too, as it gives a warning for that too.

DISCLAIMER: I put x_out.mem files in quartus project to make them appear after modelsim testbench run in quartus to make instructor's life easier. They are initially all empty!!! Output files doesn't overwrite input files!

DISCLAIMER: 256KB and 16KB memory and instruction were taking too long time to synthesize. They are easily changeable. Follow the comment instructions at the relevant places. They are currently both 1KB.